Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

Generating Microservice Applications for Performance Benchmarking

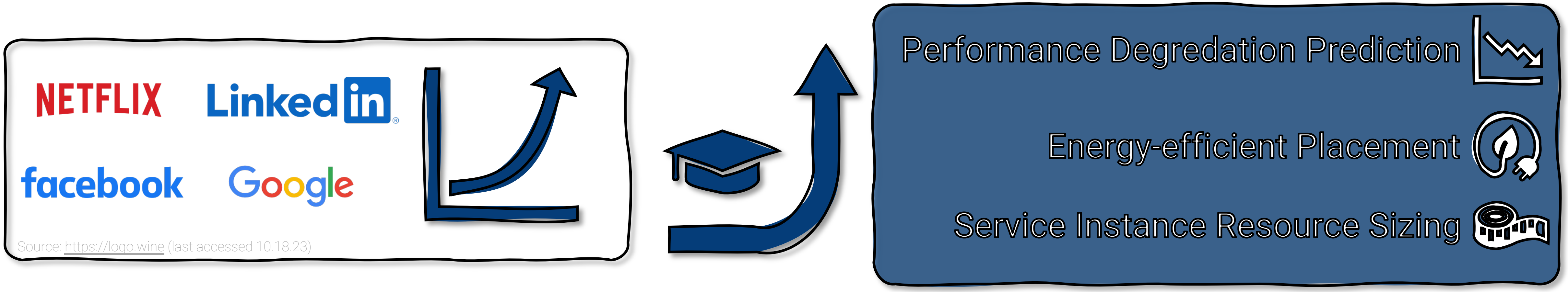
Symposium on Software Performance 2023

Yannik Lubas

07.11.23

https://se.informatik.uni-wuerzburg.de

Microservices in the Academia



Source: https://logo.wine (last accessed 10.18.23)

Performance Degredation Prediction

Energy-efficient Placement

Service Instance Resource Sizing

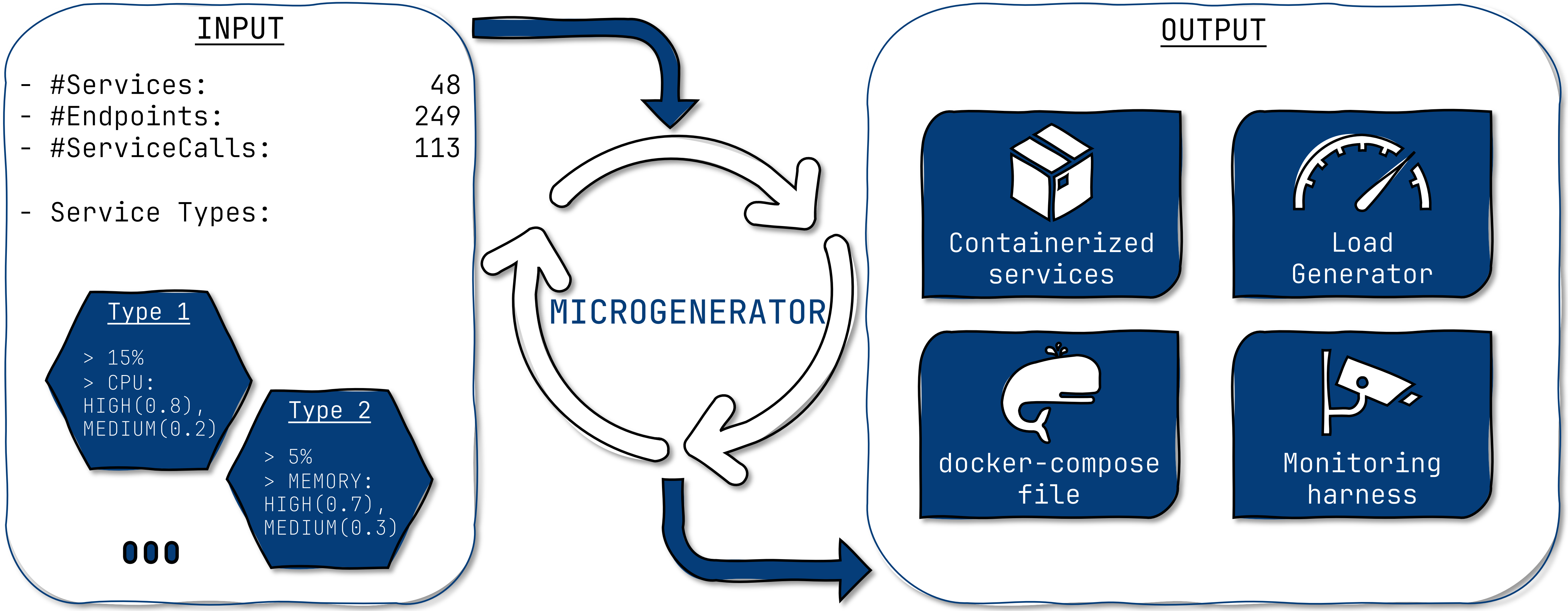
- Few open-source microservice applications suited for performance benchmarking
- Complicated and error-prone benchmarking setup
- Reference applications require application specific knowledge

Solution Proposal: MicroGenerator

✓

- Generation of microservices with configurable performance characteristics (i.e. CPU-intensive)
- Composition of the generated microservices into deployable microservice applications
- Mostly automated benchmarking process using a pre-configured monitoring harness

Generation Workflow



INPUT

- #Services: 48
- #Endpoints: 249
- #ServiceCalls: 113

- Service Types:

Type 1

> 15%
> CPU:
HIGH(0.8),
MEDIUM(0.2)

Type 2

> 5%
> MEMORY:
HIGH(0.7),
MEDIUM(0.3)

MICROGENERATOR

OUTPUT

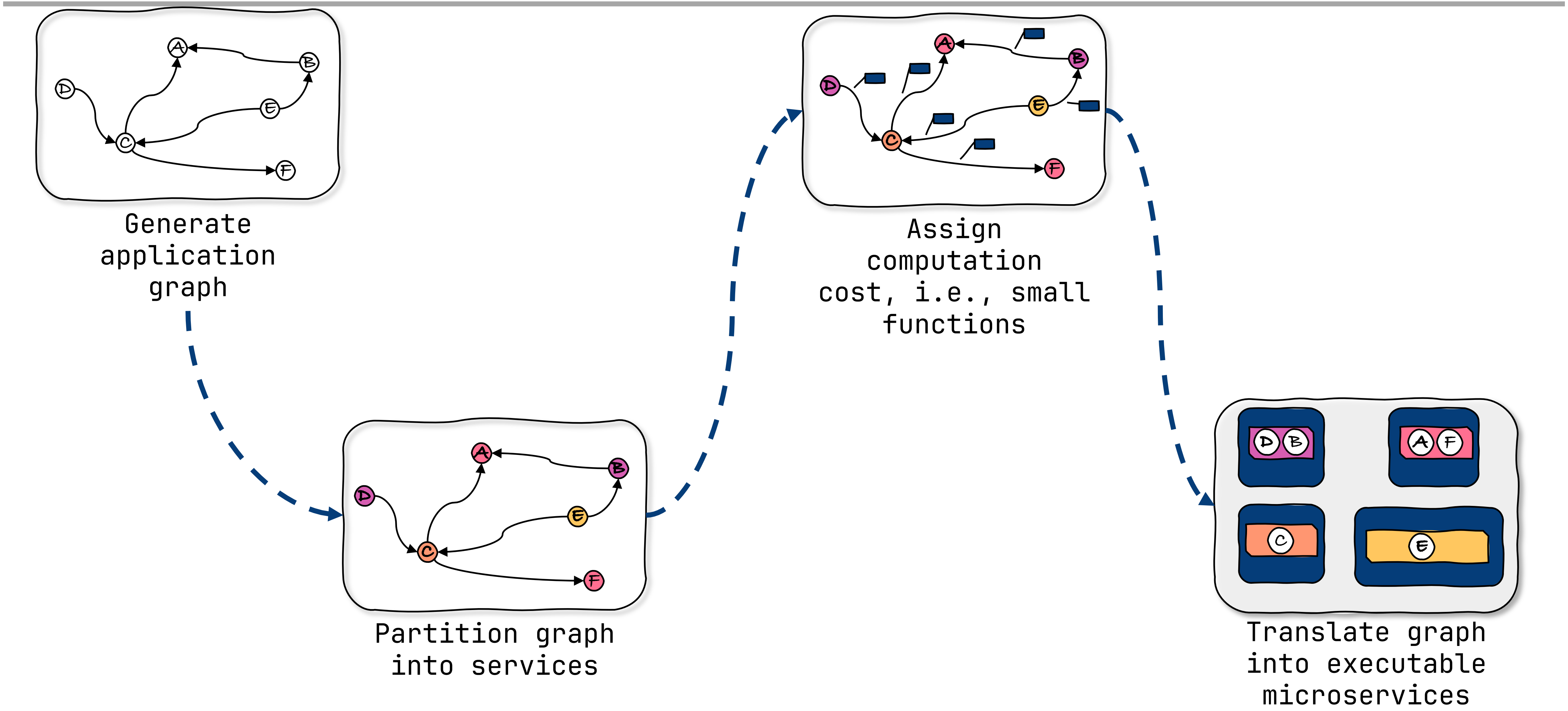
Containerized services

Load Generator

docker-compose file

Monitoring harness

Generating Microservices



Generate application graph

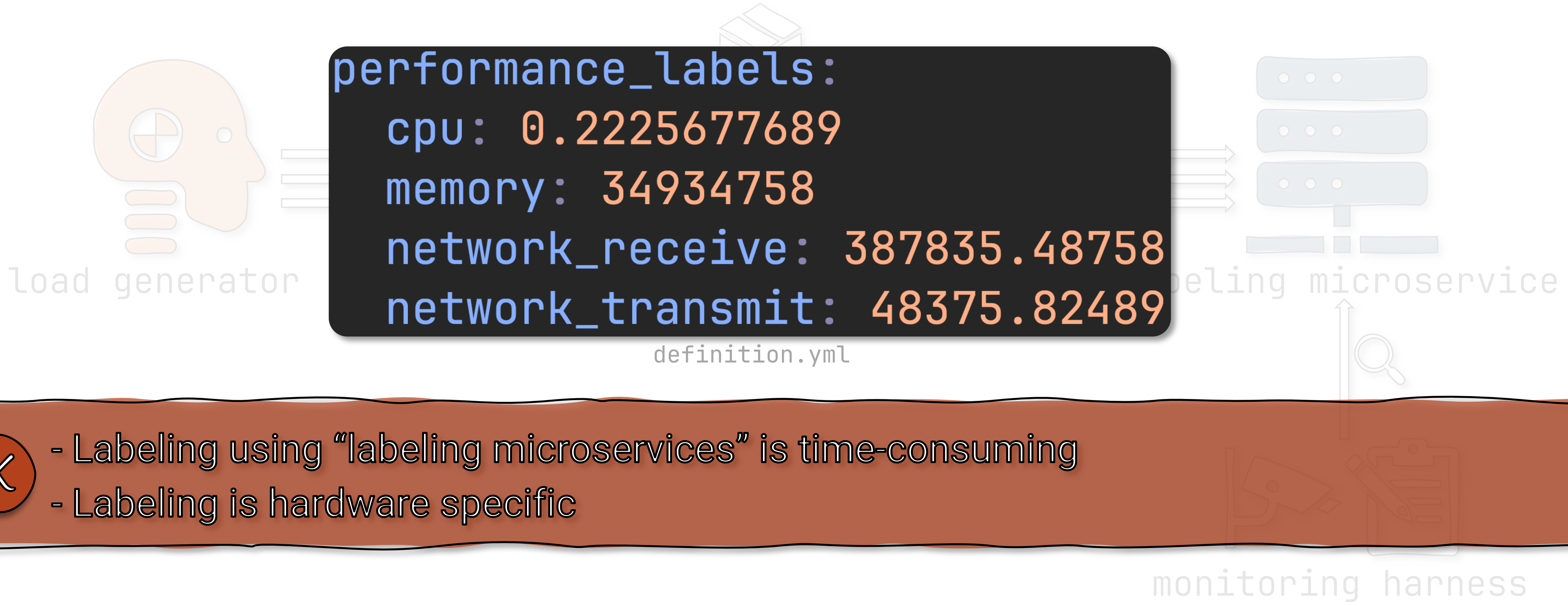
Partition graph into services

Assign computation cost, i.e., small functions

Translate graph into executable microservices

Operation Performance Labels

- "Labeling microservice" with *one* operation
- Repeat benchmark at various load levels to obtain performance measurements



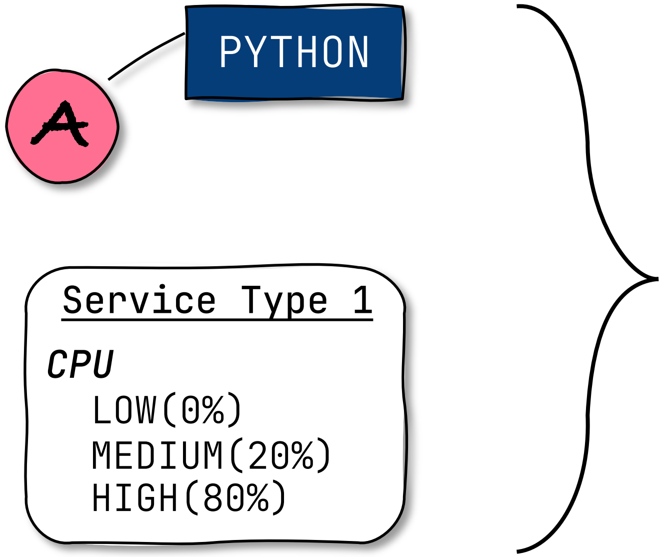
```
performance_labels:
  cpu: 0.2225677689
  memory: 34934758
  network_receive: 387835.48758
  network_transmit: 48375.82489
```

definition.yml

- Labeling using "labeling microservices" is time-consuming
- Labeling is hardware specific

Performance Characteristics

- Define performance characteristics on the endpoint-level per service type
- Use three bins per performance label (LOW, MEDIUM, HIGH)



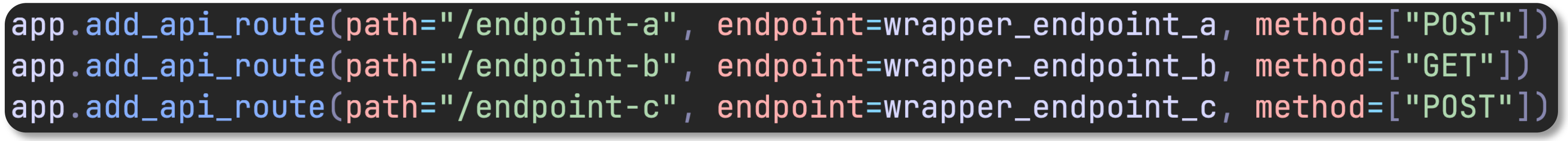
Service Type 1

CPU
LOW(0%)
MEDIUM(20%)
HIGH(80%)

Operation	CPU
generate_random_number	0.225

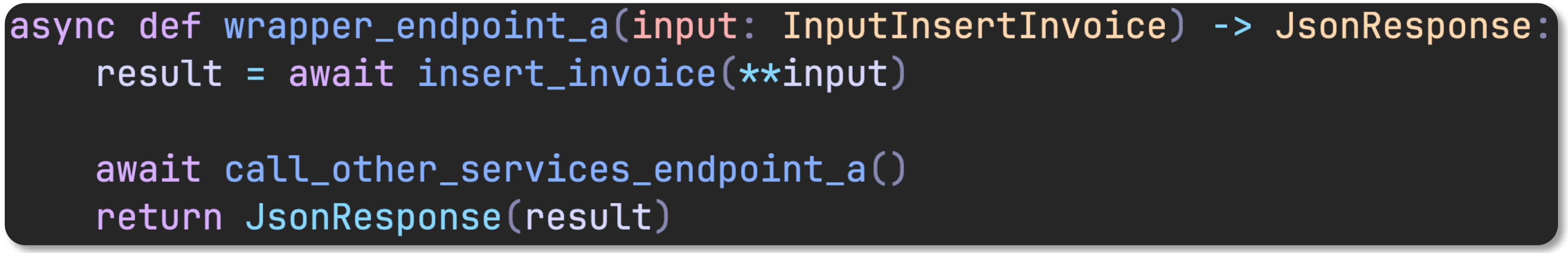
Operation	CPU
update_invoice	0.770
insert_invoice	1.026
update_invoice	0.770
insert_invoice	1.026

Generated Server File



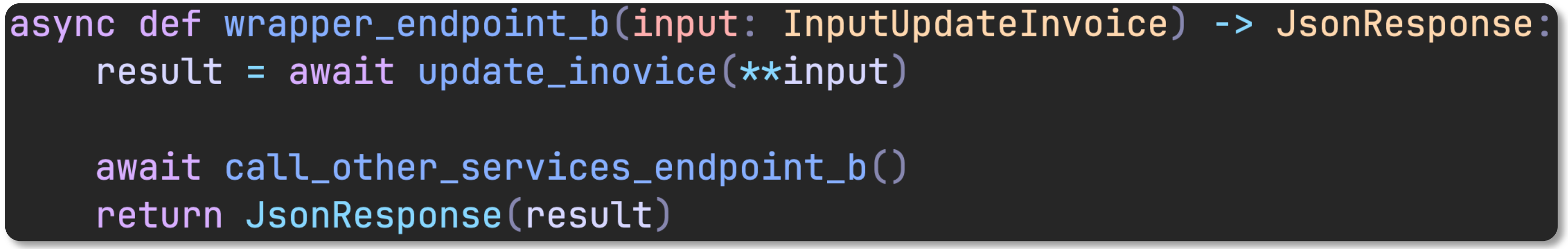
```
app.add_api_route(path="/endpoint-a", endpoint=wrapper_endpoint_a, method=["POST"])
app.add_api_route(path="/endpoint-b", endpoint=wrapper_endpoint_b, method=["GET"])
app.add_api_route(path="/endpoint-c", endpoint=wrapper_endpoint_c, method=["POST"])
```

main.py



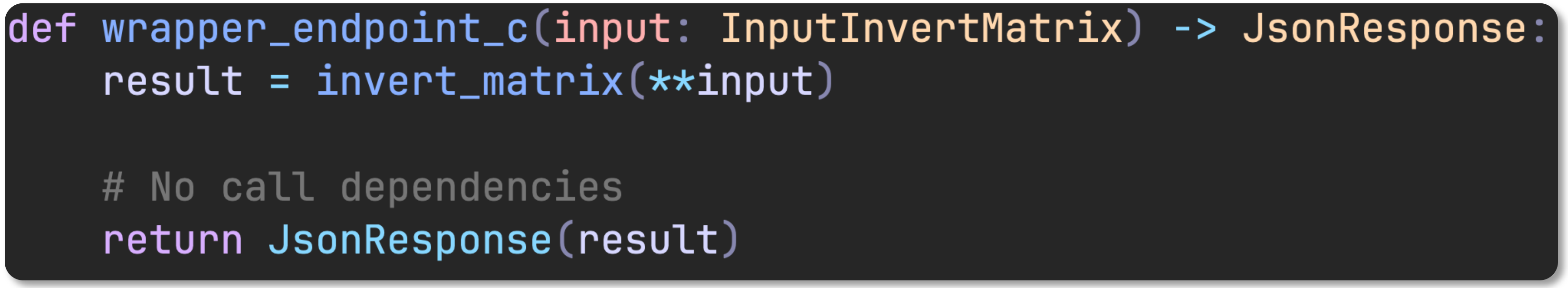
```
async def wrapper_endpoint_a(input: InputInsertInvoice) -> JsonResponse:
    result = await insert_invoice(**input)

    await call_other_services_endpoint_a()
    return JsonResponse(result)
```



```
async def wrapper_endpoint_b(input: InputUpdateInvoice) -> JsonResponse:
    result = await update_inovice(**input)

    await call_other_services_endpoint_b()
    return JsonResponse(result)
```



```
def wrapper_endpoint_c(input: InputInvertMatrix) -> JsonResponse:
    result = invert_matrix(**input)

    # No call dependencies
    return JsonResponse(result)
```

Evaluation

1. Operation Selection Algorithm
2. Configuration of performance characteristics
3. Useability of generated applications as training data

Use case study in the domain of resource saturation detection classifiers

✓

- Monitorless [1]: Random Forest classifier using platform-level metrics
- Multiple training datasets comprising performance data from generated and “real-world” applications
- Test dataset only contains data from a “real-world” application

[1] J. Grohmann, P. K. Nicholson, J. O. Iglesias, S. Kounev, and D. Lugones, "Mon- itorless: Predicting Performance Degradation in Cloud Applications with Machine Learning," in Proceedings of the 20th International Middleware Conference, 2019, pp. 149–162.

Evaluation Setup

9 Application Datasets

Apache Solr (Sol):
Web Search Index

Memcached (Mem):
In-memory Object Store

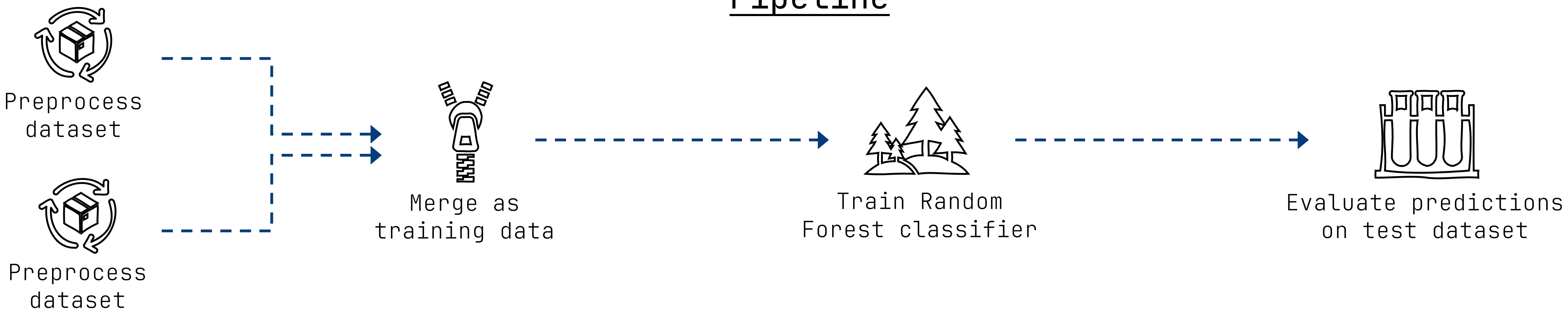
Cassandra (Cas):
Document Database

App1 - App6:
Network + CPU heavy
generated applications

Test Dataset

TeaStore (Tea):
E-commerce microservice
application

Pipeline



Preprocess
dataset

Preprocess
dataset

Merge as
training data

Train Random
Forest classifier

Evaluate predictions
on test dataset

Some Evaluation Results

- Absolute prediction performance not relevant
- Prediction performance indicates the ability to represent TeaStore

Training Data	Accuracy (%)	F1 Score (%)
Sol	94.2	88.6
App2+App3	**97.0**	**93.5**

- Generated applications exhibit configured performance characteristics
- Performance measurements of generated applications can substitute measurements from existing reference applications

Conclusion

Difficult to obtain performance measurements from representative, open-source microservice reference applications

Automatic generation of microservice applications with configurable performance characteristics

Facilitate mostly automatic performance benchmarking of the generated applications

✓ Generate microservice applications with virtually unlimited scope

Easily collect performance benchmarking data

Use *operations* and graph generation for generating executable microservices