# A Reproducible Tutorial on Reproducibility in Database Systems Research

VLDB 2024

Denis Hirn
{denis.hirn, tim.fischer}@uni-tuebingen.de
University of Tübingen

Tim Fischer

Gökhan Kul
gkul@umassd.edu
University of Massachusetts

# Overview

Part 1

Reproducibility Definition and Goals

Part 2

Containerization Technologies

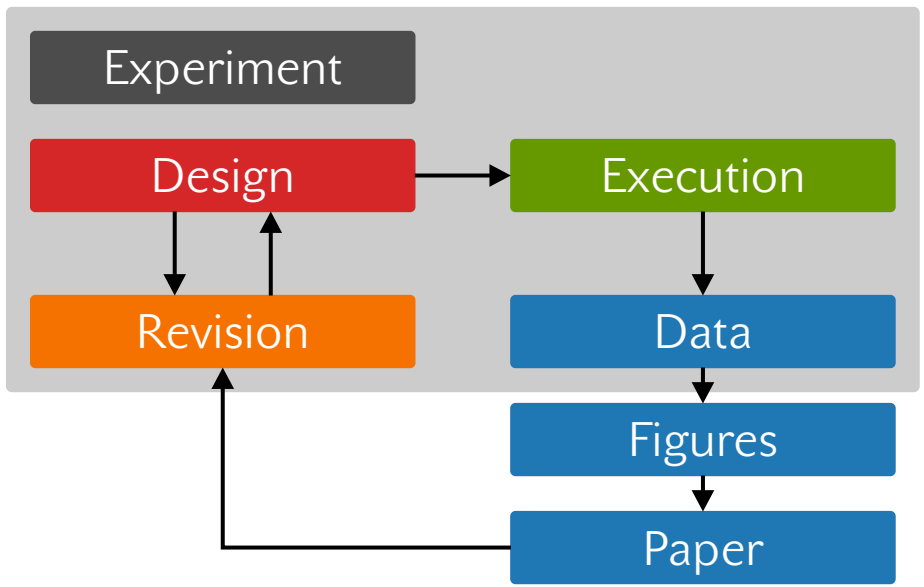
Part 3

Automation

Part 4

Documentation

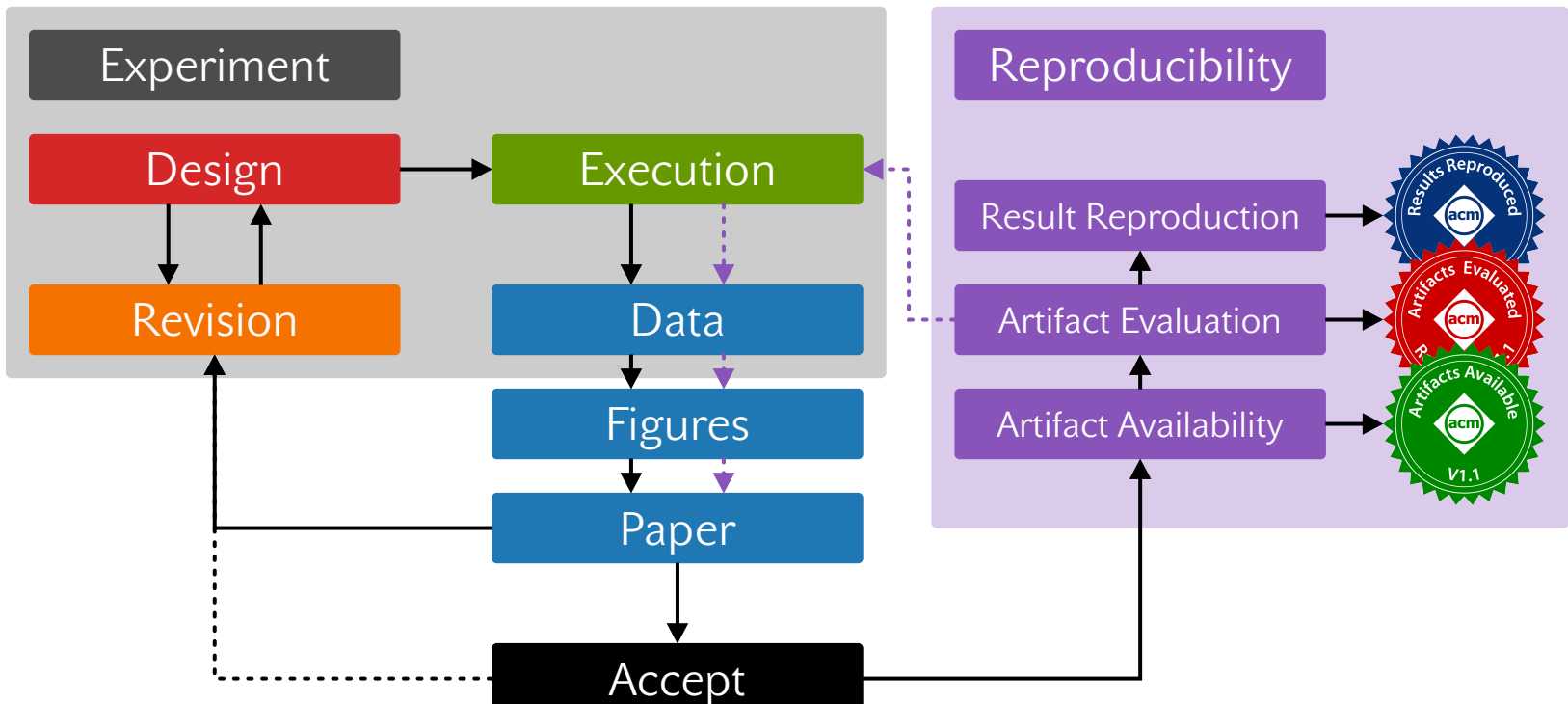
# Reproducibility Workflow in Database Systems Research




SIGMOD reproducibility effort: https://reproducibility.sigmod.org

VLDB reproducibility effort: https://vldb.org/pvldb/reproducibility

# Reproducibility Workflow in Database Systems Research




SIGMOD reproducibility effort: https://reproducibility.sigmod.org

VLDB reproducibility effort: https://vldb.org/pvldb/reproducibility

# Goals of Reproducibility

## Availability

- The artifacts (software, data, and documentation) should be available.

## Reproducibility

- Central results and claims should be supported.
- Experimental results can be independently verified.
- Exact replication is not always possible, but the overall results should match conclusions.

> In short: More impact, more trust, more credibility.

# Reproducibility in Practice

Reproducibility is…

- the ability to **re-run** an experiment and obtain the **same** results,
- a cornerstone of the scientific method,
- a key factor in the *credibility* of research results.

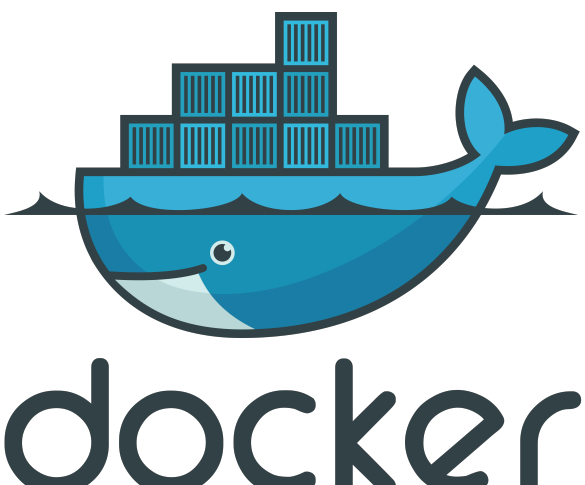
---

However…

- it can be difficult to achieve in practice,
- especially in database systems research,
- and when done as an afterthought.

The key to reproducible experiments is to design them with reproducibility in mind.

# Containerization Technologies



# Without Containers

- Ship packages: `zip`, `tar.gz`, etc.
- Manual installation of software and manual configuration.
- Dependency hell.
- “Works on my machine. `¯\_(ツ)_/¯`”
- Experiments are difficult to reproduce, because
  1. the environment (configuration) may be different,
  2. package versions may be different,
  3. the software may not be available anymore
  4. ⋮

In short, it is a mess!

# With Containers

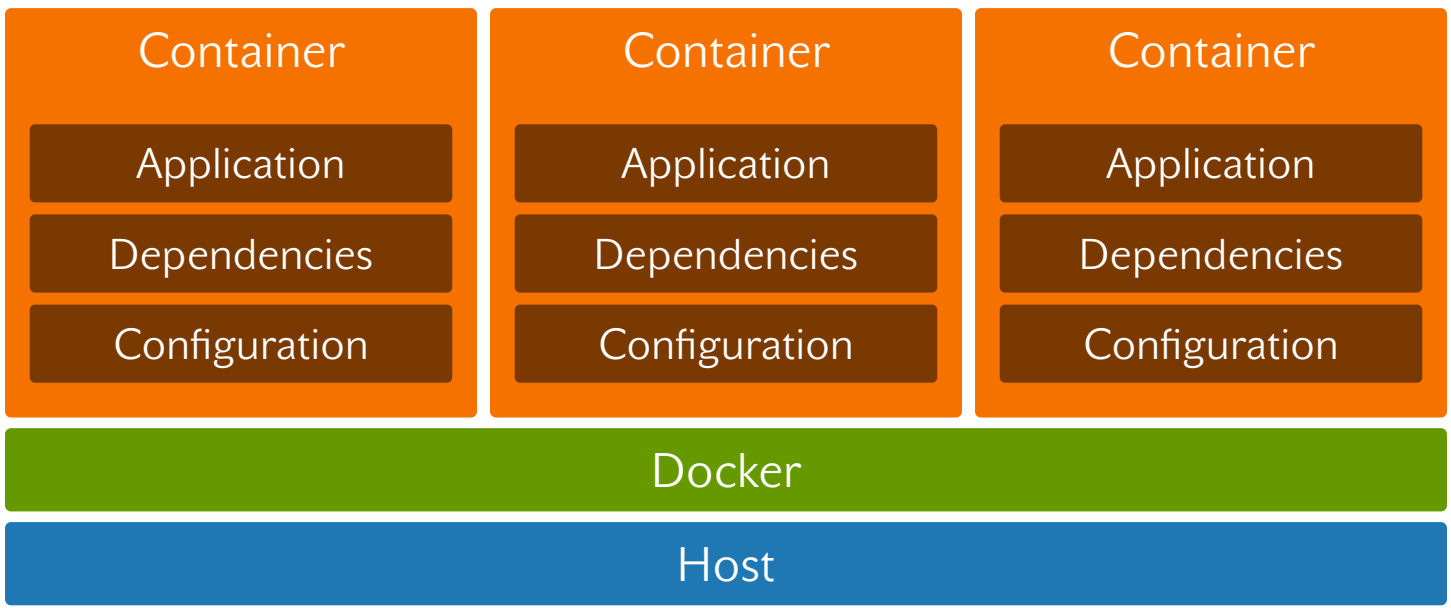
- ~~Ship packages: `zip`, `tar.gz`, etc.~~
- ~~Manual installation of software and manual configuration.~~
- ~~Dependency hell.~~
- ~~“Works on my machine. ¯_(ツ)_/¯”~~
- ~~Experiments are difficult to reproduce, because~~

---

- Containers contain **everything**. No more packages!
- Isolation.
- Works everywhere.
- Remains functional in the future.
- Experiments are easy to reproduce. For future you, and others!

Reproducibility is not a *one-time* effort.

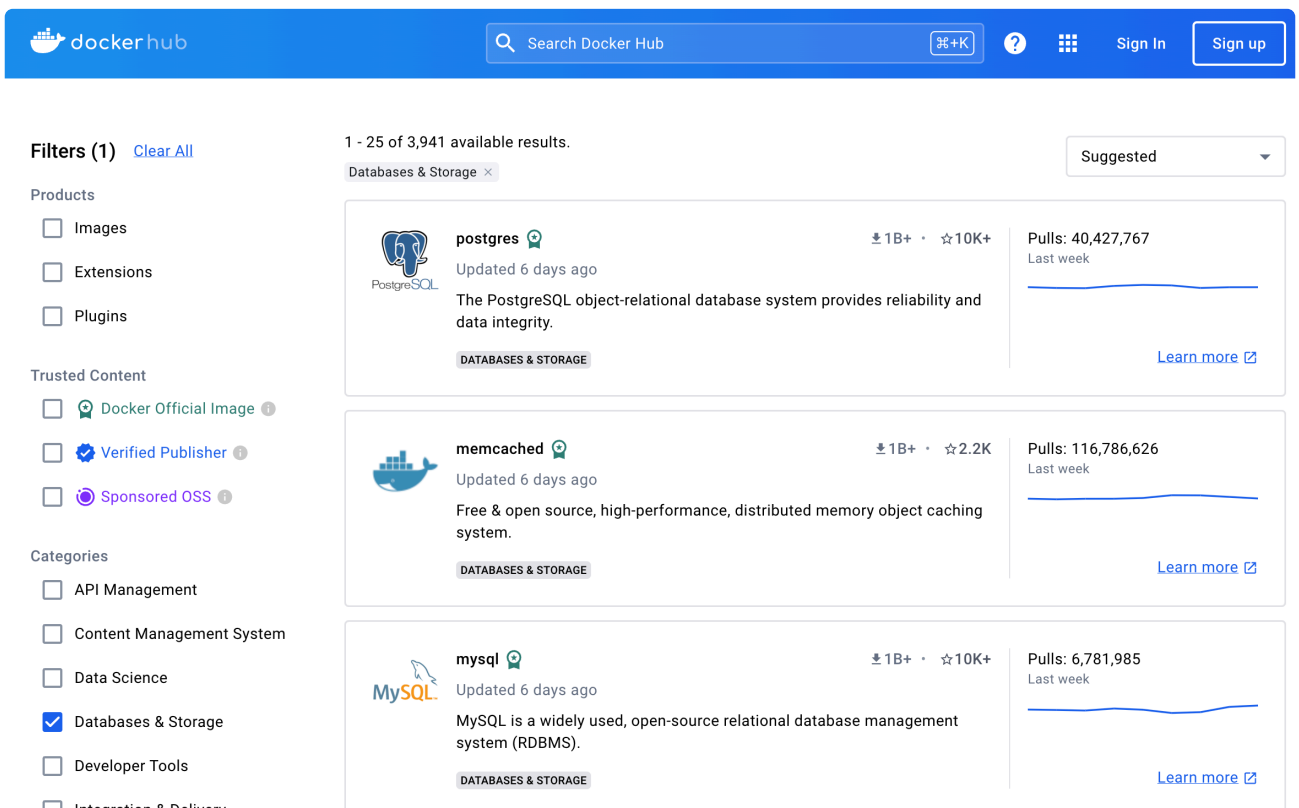
# Docker Abstraction Layers




- The containers are **isolated** from the host system and other containers.
- Eliminate the *"works on my machine"* problem.
- Prevents pollution of the host system.

# Docker Hub

- Huge repository of pre-built Docker images.
- Can be used to host your own images (aka. reproducibility artifacts).



# Dockerfiles and Docker Images

01_simple/Dockerfile.duckdb

```
FROM ubuntu:22.04 AS duckdb

RUN apt-get update && apt-get install -y \
    build-essential cmake git

RUN git clone --depth 1 --branch v1.0.0 \
        https://github.com/duckdb/duckdb.git

WORKDIR /duckdb
RUN make release -j

# Add the binary to the PATH
ENV PATH="/duckdb/build/release:${PATH}"
# Define the entrypoint
ENTRYPOINT ["duckdb"]
```

```
❯ docker build -t duckdb .
```

# Docker Compose

01_simple/docker-compose.yml

```yaml
services:
  duckdb:
    image: duckdb
    build:
      context: .
      dockerfile: Dockerfile
  postgres:
    image: postgres:16
```

```
❯ docker compose up -d
```

- Docker Compose is used for multi-container applications.
- Single command to `start` | `stop` | `remove` all services.
- Simplifies wiring up the containers and volumes.

# Docker Volumes

- **Volumes** are a way to persist data in Docker containers.
- Can be used to share data between containers.
- Bind mounts can be used to share data between the host and the container.

01_simple/docker-compose.yml

```yaml
services:
  duckdb:
    image: duckdb
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - type: bind
        source: ./data
        target: /data
  postgres:
    image: postgres:16
    volumes:
      - postgres_volume:/var/lib/postgresql/data
      - type: bind
        source: ./data
        target: /data
```

```
❯ docker compose up -d
```

# Takeaways from Containerization

1. Containers provide consistent software environments for experiments.
2. Images can be shared, reused, and versioned.
3. Containers eliminate manual software installation on the host machine.

# Automation


GNU Make

# Automate as Much as Possible

- Automation is key to reproducibility.
- Experimental execution should be as simple as possible.
- The user should not have to worry about the details.
- The user should be able to reproduce the results with a single command.
- Post-processing should be automated, too.

# Experimental Scenario

- Run a set of experiments (*e.g.*, TPC-H queries)
  1. On DuckDB
  2. On PostgreSQL
- Goal is to compare the performance of the two systems.

# Automation Script

01_simple/run.sh

```
mkdir -p ./data

docker compose build

docker compose up postgres -d --wait

docker compose run --rm duckdb bash "scripts/generate-tpch-duckdb.sh"

docker compose run --rm duckdb bash "scripts/run-experiments-duckdb.sh"

docker compose exec postgres bash "scripts/load-tpch-postgres.sh"

docker compose exec postgres bash "scripts/run-experiments-postgres.sh"

docker compose down -v --rmi all
```

- Single “entrypoint” for the user.
- Include setup, execution, and cleanup.

# Auditability

```
#? ▉
⋮
⏱ however long later
⋮
Done.
❯ ▉
```

No indication of progress, about what is happening, or how long it will take.

```
#? ▉
⋮
Running experiment 1...
...
3 / 5:
⋮
Done.
❯ ▉
```

Output the progress of the experiments to the console.

- Auditability is the ability to trace the execution of a script.
- Scripts should provide feedback to the user.
- Progress indicators can be helpful.
- Log files should be timestamped and written to disk.
- Estimating the time to completion can be helpful.

# Dockerized TeX Compilation

02_intermediate/docker-compose.yml

```yaml
services:
  # ...
  latex:
    image: danteev/texlive
    volumes:
      - type: bind
        source: .
        target: /project
    working_dir: /project
```

---

02_intermediate/scripts/build-paper.sh

```sh
#!/bin/bash

mkdir -p /project/data/paper
pushd /project/paper > /dev/null
latexmk\
  -outdir=/project/data/paper \
  -pdf \
  -interaction=nonstopmode \
  -shell-escape \
  -synctex=1 \
  -f \
  paper.tex
popd > /dev/null
```

# Automation using a Build System: `make`

03_advanced/Makefile

```
.PHONY: docker
docker: # Build and start Docker services
  ...

.PHONY: setup
setup: docker # Setup the environment
  ...

.PHONY: experiments
experiments: setup # Run the experiments
  ...

.PHONY: paper
paper: experiments # Build the paper
  ...

.PHONY: clean
clean: # Clean up everything
  ...
```

- Use `make` to automate the entire workflow.
- Each target depends on the previous target.
- The `clean` target cleans up everything.

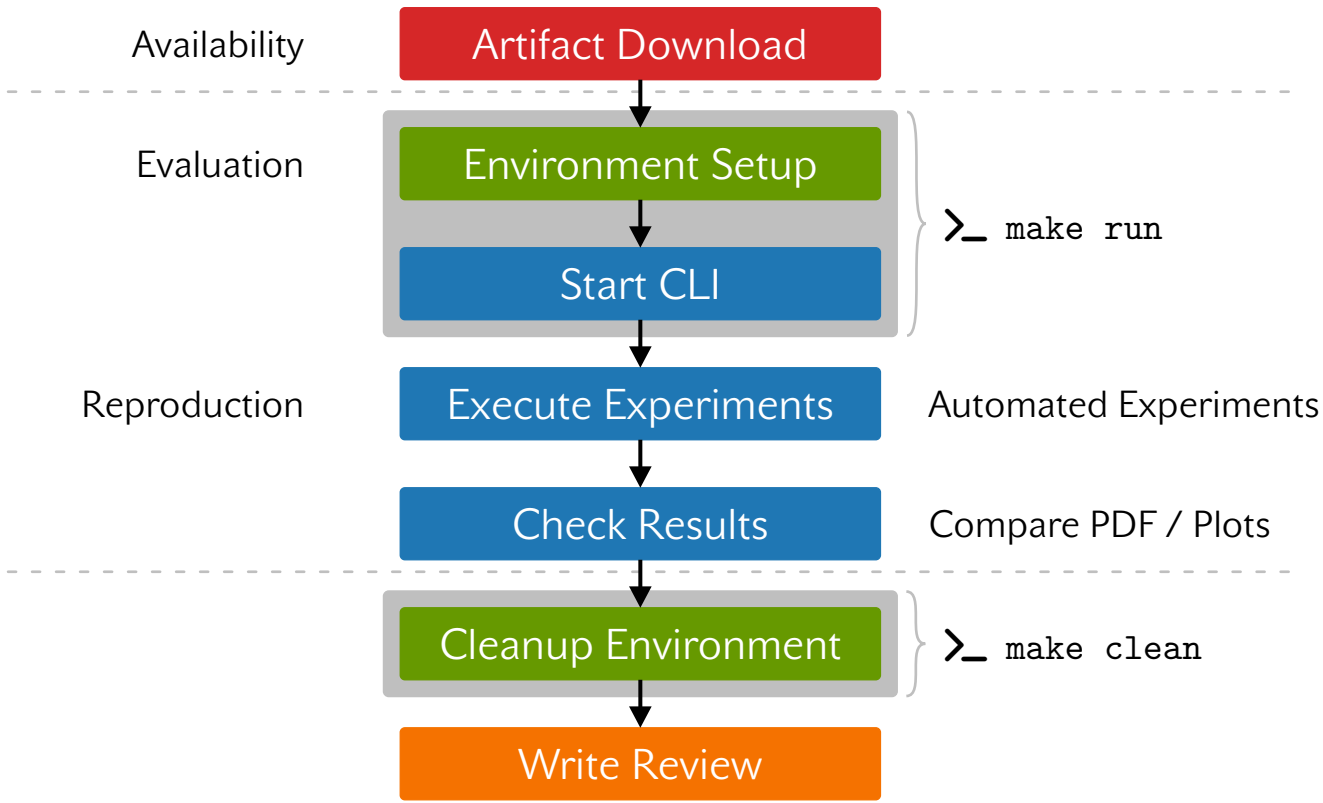
# Use Docker within Docker

04_full/docker-compose.yml

```yaml
services:
  docker:
    image: runner
    privileged: true
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./project:/project
    working_dir: /project
```

- Docker in Docker is a common pattern for CI/CD pipelines.
- Prevents polluting the host system.
- Gives scripts full control over the Docker environment, without needing to rely on the host system's environment.
- Use the `privileged` flag to run Docker within Docker.

# Reproducibility Review Workflow

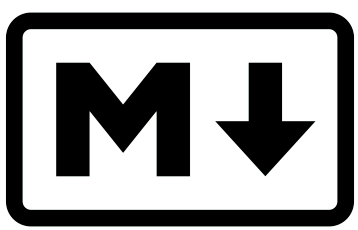



The review process should be as easy and automated as possible.
It is the authors' responsibility to provide an easily reproducible artifact.

# Takeaways from Automation

1. Automation is key to reproducibility.
2. Be mindful of the reviewer's time and effort.
3. The benefits of automation will also apply to you as the author!

# Documentation



# Artifact Documentation

03_advanced/README.md

````
# Overview
This repository contains the artifacts for the tutorial
"A Reproducible Tutorial on Reproducibility
 in Database Systems Research".

## Software Requirements
The artifact requires Docker and Docker Compose.

## Hardware Requirements
At least 4GB of memory and 2 CPU cores are required.

## Time Requirements
* Getting Started: 5 minutes
* Building the Docker image: ~20 minutes
* Running the experiments: ~5 minute

## Reproducibility
To reproduce the experiments, follow these steps:
* Clone the repository
* In a terminal, navigate to the repository and run the following command:
  ```bash
  make run
  ```
* The experiments will run and the results will be stored in the `data` directory.

## Cleanup
To cleanup, run the following command:
```bash
make clean
```
````

- Clearly mention side effects of commands.
- Do not assume the user has any knowledge.
- Provide a step-by-step guide.
- Estimate the human time required for each step.
- Provide a cleanup script.
- ⚠ Do **not** break the user's system.

# Concluding Remarks

# Limitations of Containerization

- Not all software can be containerized.
- Not all workflows can be automated.
- Containerization provides the software environment, but not the hardware environment.
- Experiments may require specialized hardware.

# Long-running Experiments, Specialized or Massive Hardware Requirements

- Few reviewers have a supercomputer at hand.
- Experiment should be executable on regular hardware (*e.g.*, on a laptop).
- Provide reviewers with a downscaled version of the experiment.
  1. Use a smaller dataset,
  2. fewer iterations,
  3. fewer parameters.
- The main conclusions should be reproducible.

If not possible (which should be a rare exception), contact the program chairs for advice.

# Cloud Deployment

- https://docs.cloudlab.us/repeatable-research.html
- https://www.chameleoncloud.org/blog/2021/11/22/using-chameleon-for-artifact-evaluation/

# A Reproducible Tutorial on Reproducibility in Database Systems Research

VLDB 2024

Denis Hirn
{denis.hirn, tim.fischer}@uni-tuebingen.de
University of Tübingen

Tim Fischer

Gökhan Kul
gkul@umassd.edu
University of Massachusetts