# ProfiT-HPC: Architecture and Recommendation System

Azat Khuziyakhmetov

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Performance-Engineering Workshop (TU Dresden)
25.-26.03.2019

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

# Overview

# Organization


- ProfiT-HPC: Profiling Toolkit for High Performance Computing
- Duration: 01.02.2017 - 31.01.2020
- Partners:





# Goals

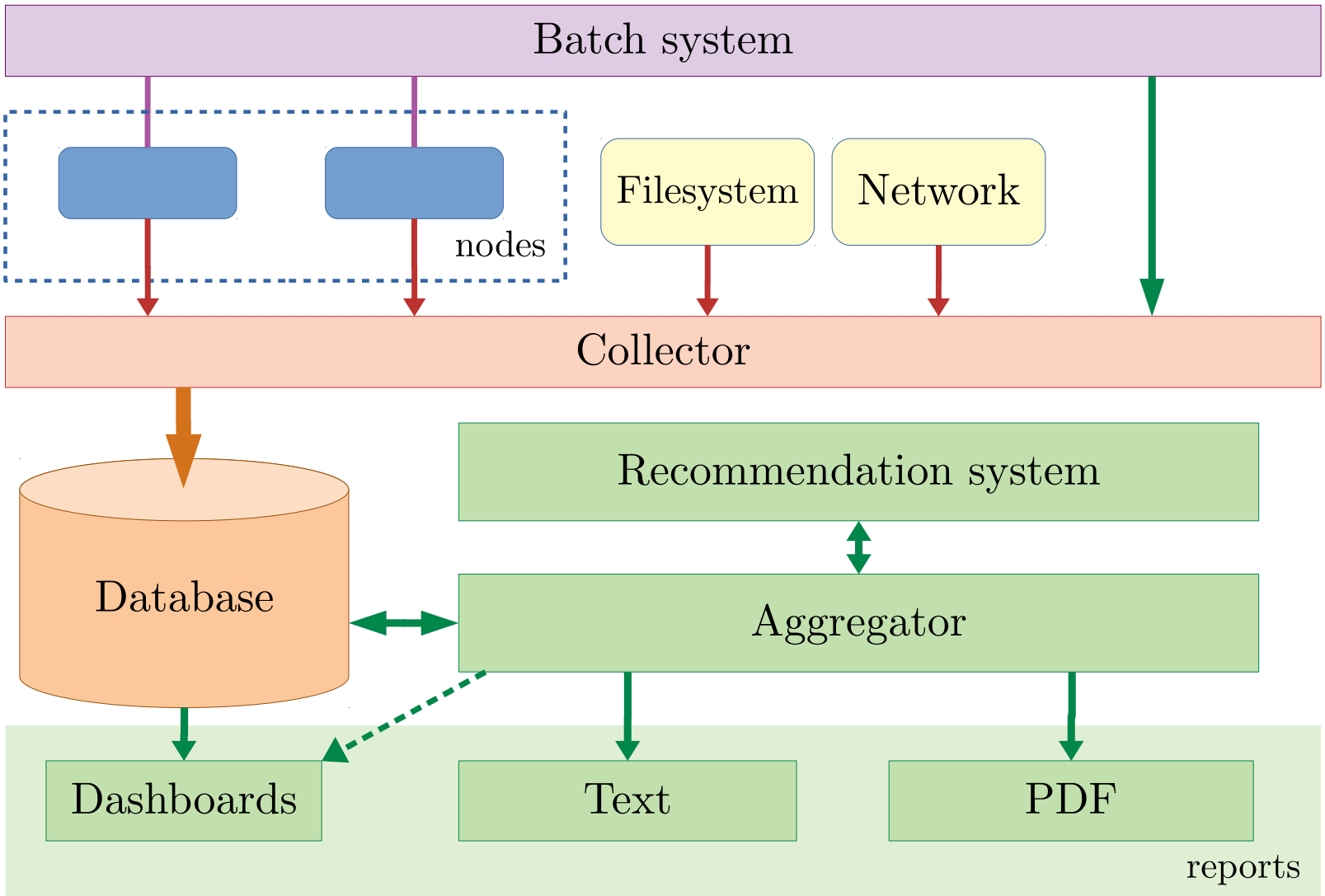

## Motivation

Comprehensive performance analysis of jobs.
Users get reports with necessary information about the job to evaluate its efficiency and tweak submission parameters or programs if needed.

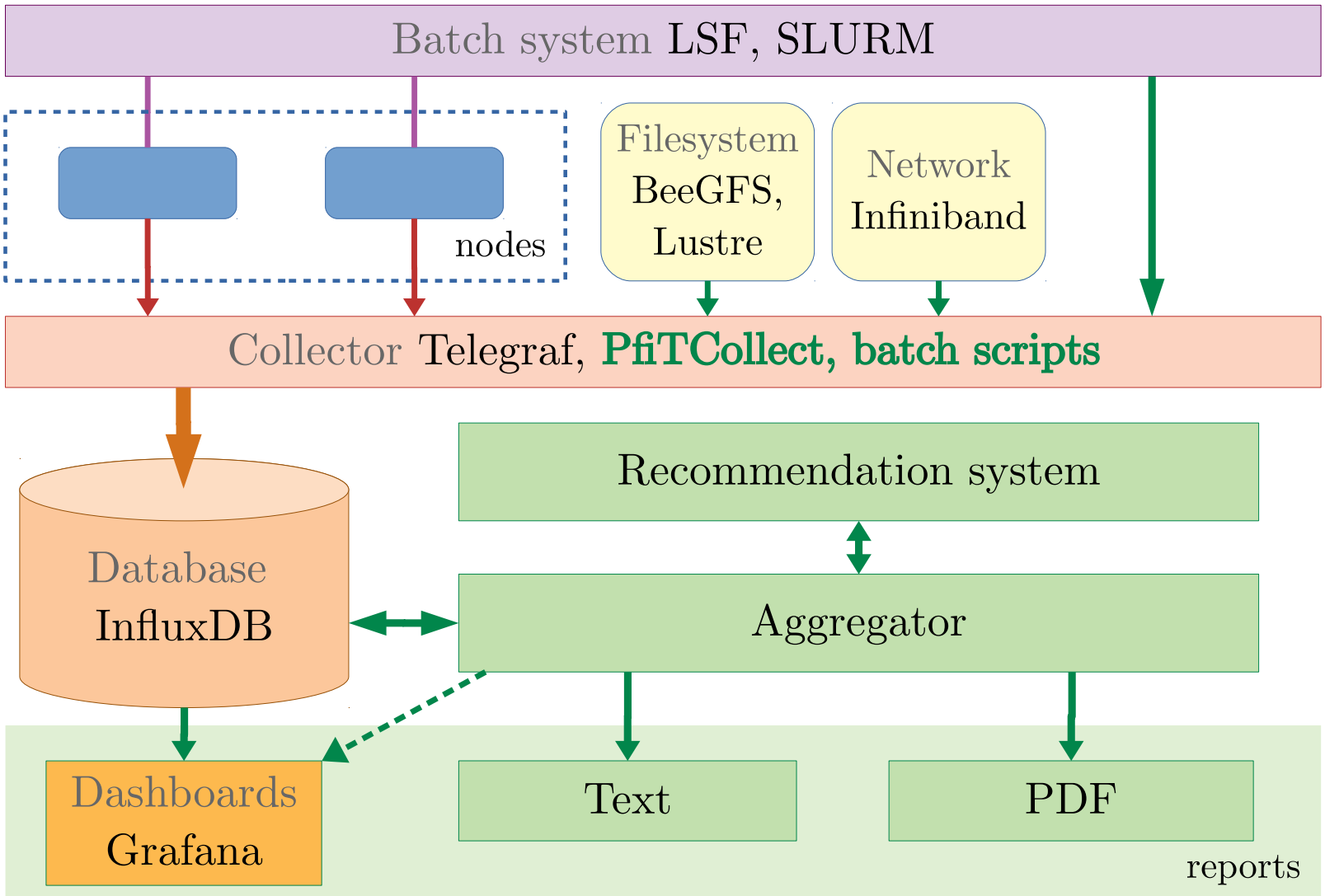
## Goals

- Auto generated reports with following data:
  - chosen metrics and performance indicators
  - recommendations to reach higher efficiency
- Improved resources utilization of HPC

# Architecture



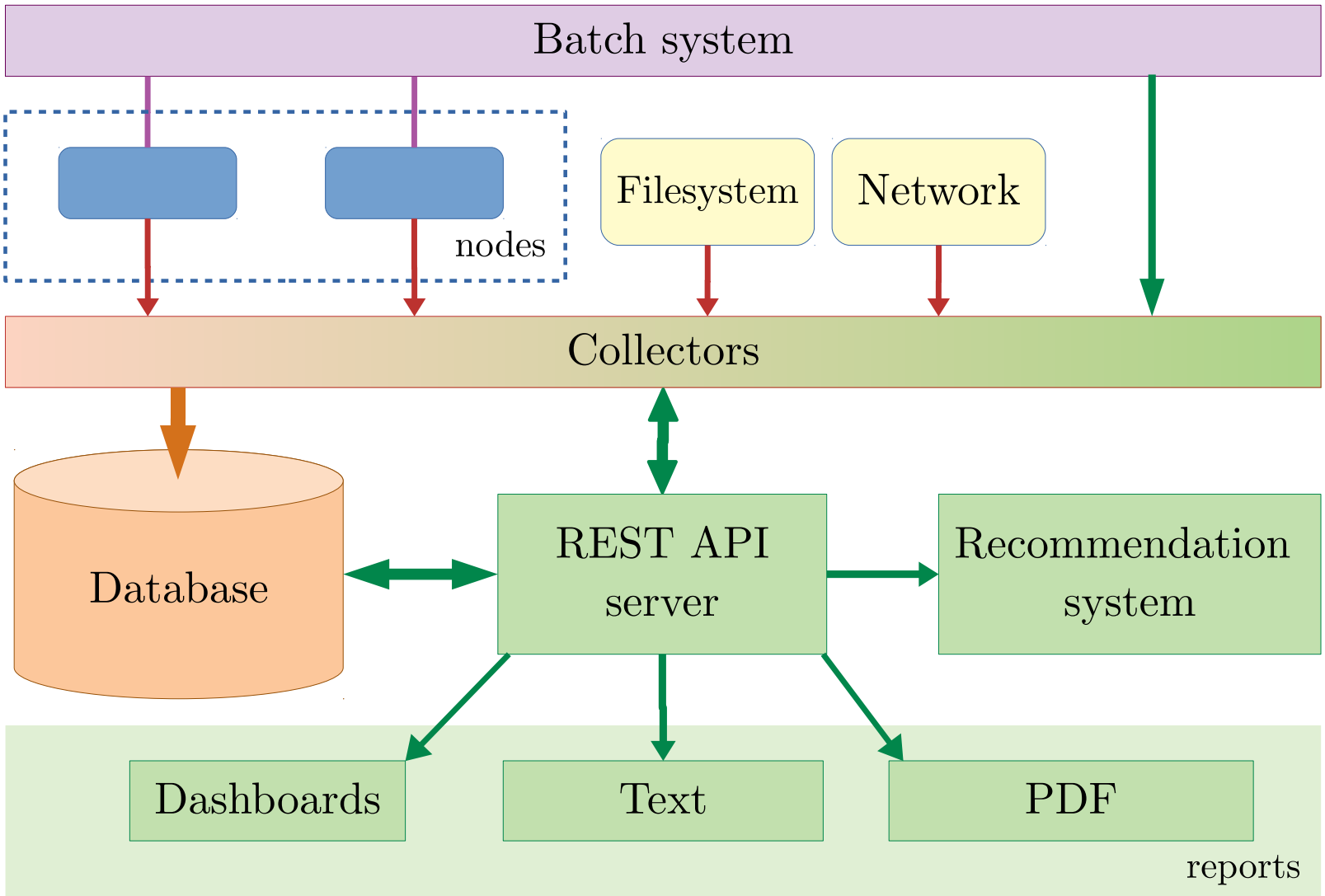
# Architecture Implementation



# Implementation

| Module | Language |
|---|---|
| PfiTCollect | *C* |
| Telegraf plugins | *GO, bash* |
| Grafana dashboards | *InfluxQL* |
| Aggregator | *Python* |
| Text report | *Python* |
| PDF report | *Python* |
| Recommendation system | *Python* |
| Epilog scripts | *bash* |

# Alternative architecture



# Why modular?


## System design

- Every part is independent
- Communicate only via established interface (API, REST)
- Modules can be substituted / reimplemented

## Development

- Every site implements module independently
- Easy to substitute dependent modules with stub implementation
- Every module can be encapsulated in its own environment

# Dev environment


Modular architecture allows to develop modules of the toolkit with stub implementation of dependencies.

| **Module** | **Dependency** | **Stub type** |
|---|---|---|
| PfiTCollect | OS & Env | Container |
| Telegraf plugins | OS & Env | Container |
| Aggregator | Database | Container |
| Grafana dashboards | Database | Container |
| Text report | Aggregator | JSON file |
| PDF report | Aggregator | JSON file |
| Recommendation system | Aggregator | Python object |
| Epilog scripts | Batch system | Bash script |

# Metrics


## Nodes

- CPU
- Memory
- System

## Processes

- CPU times
- Memory
- IO

## Filesystem

- Lustre
- BeeGFS
- Local

## Others

- Job
- GPU
- Network (Infiniband)

# PfiTCollect


- light weight metric collector
- implemented for ProfiT to gather metrics fast, without overhead
- substitute for Telegraf and works with InfluxDB

## Telegraf

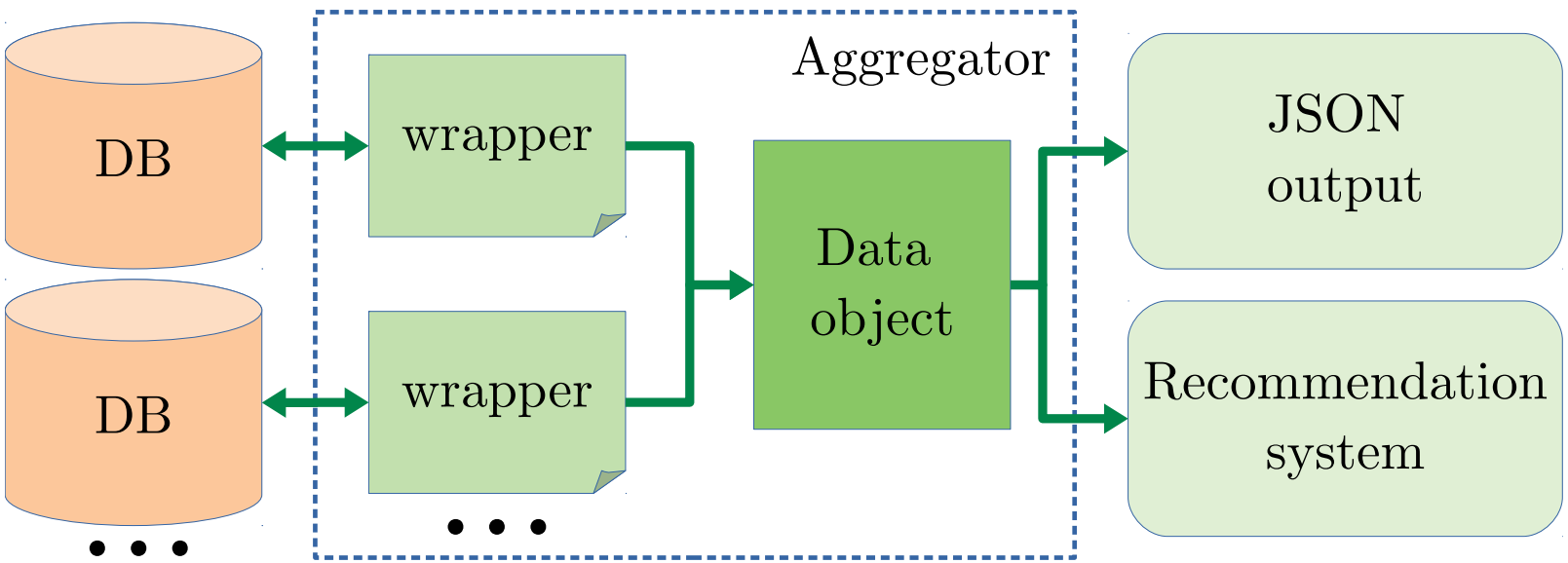
+ Large community
+ Extendable (plugins)
- Heavy (27 MB Executable)
- Many dependencies (GO)

## PfiTCollect

+ Very lightweight (∼100KB)
+ Easy compilation
- No external community

# Aggregator

- Aggregator - Python module to collect the data and pass it further to reports or other parts of the toolkit
- It also provides API to retrieve the data as a Python object



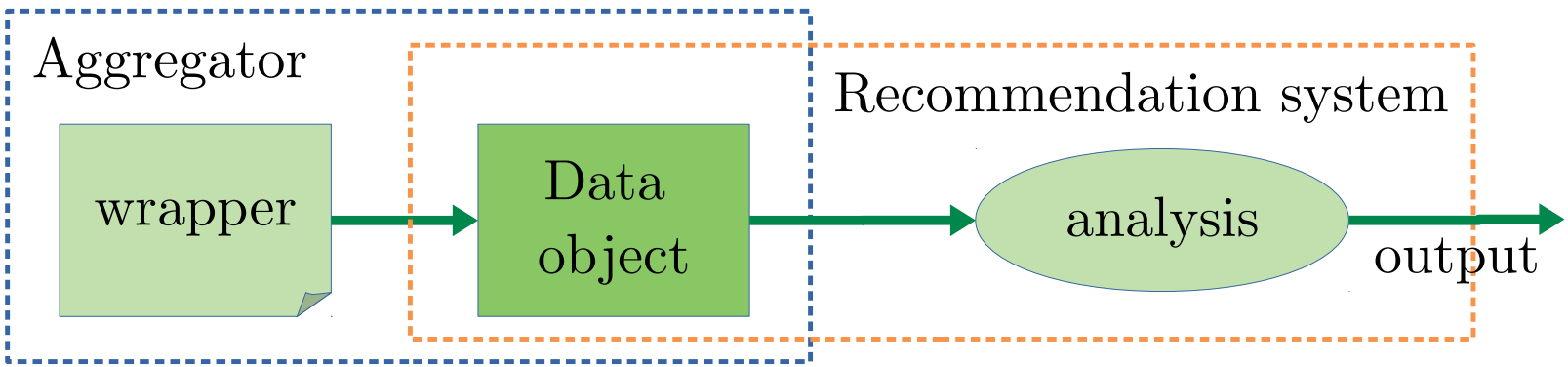
# Recommendation system

# Recommendation system

## Definition

Recommendations are comprehensive instructions for users to help with improving efficiency of their jobs

Recommendation system is designed to be completely separated from other parts of the toolkit. Only data should be provided.



# Rule based recommendations

Rules are natural way to give recommendations in current systems by experts

## Example

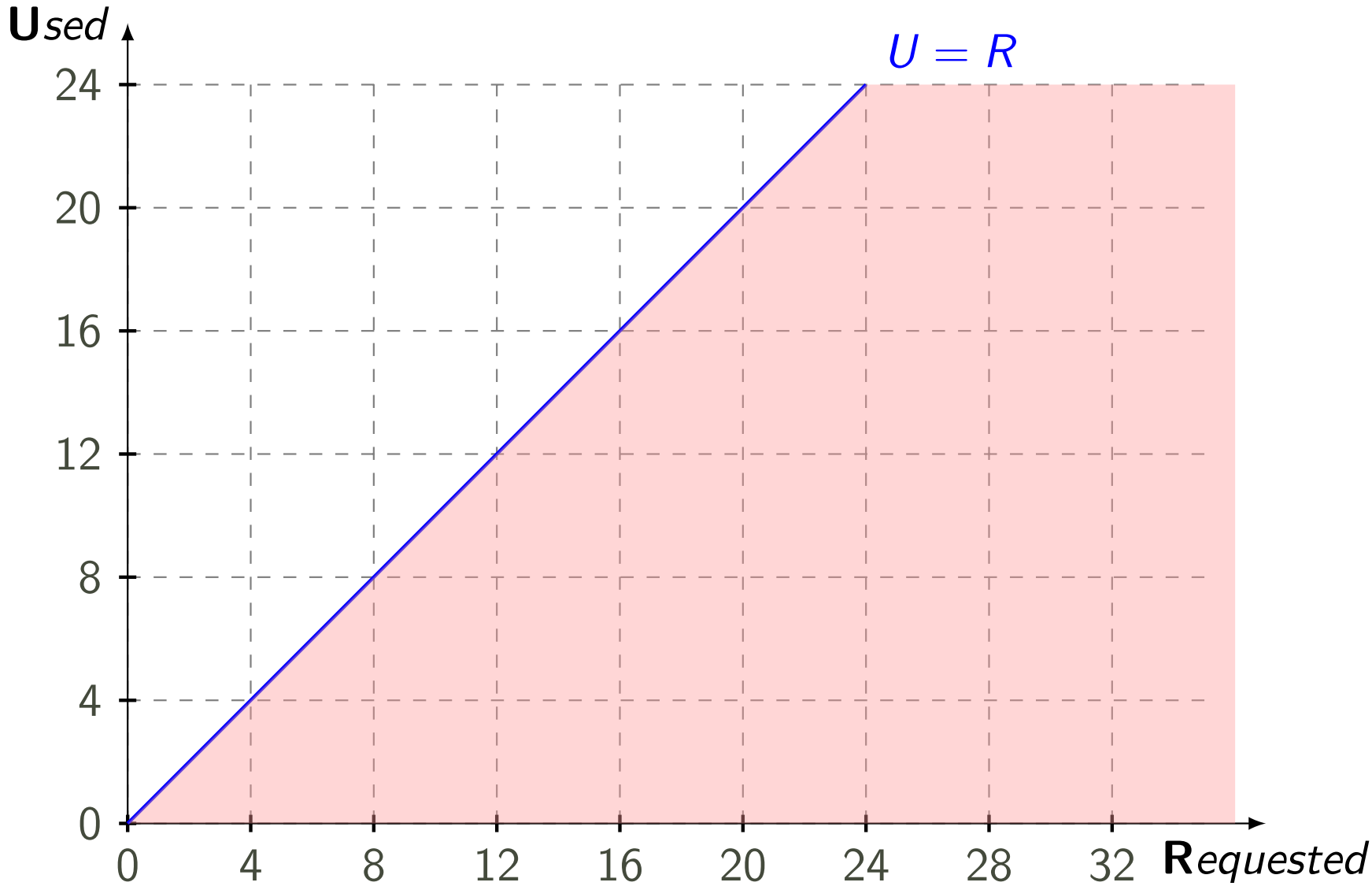
**Case** Users request high wall time, resulting to long pending state

**Rule** If runtime of the job is half or less than requested walltime ($\texttt{runtime} \leq \texttt{requested}/2$)

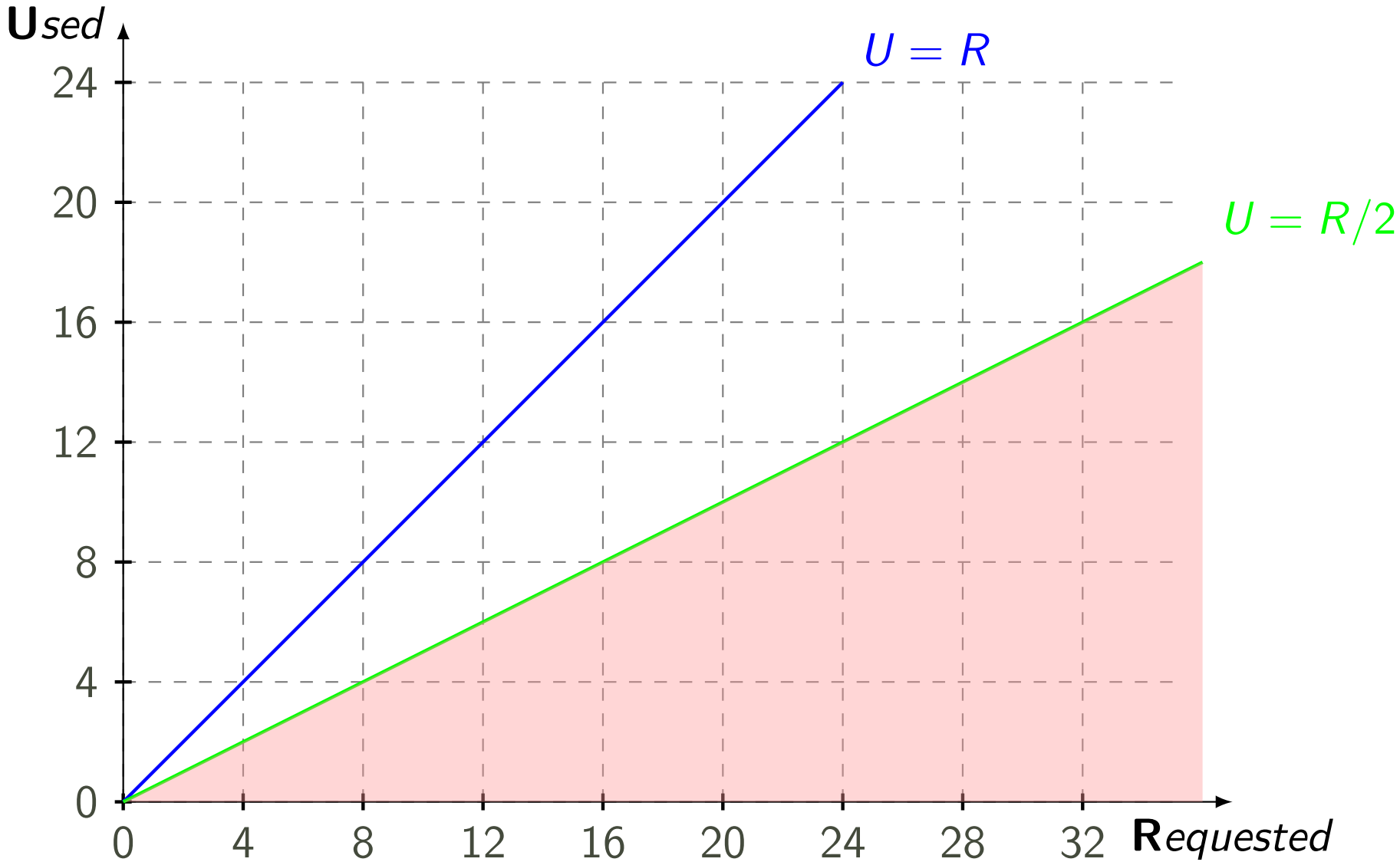
**Advice** Request less walltime

However, deriving such rules are complicated, even for such simple cases...

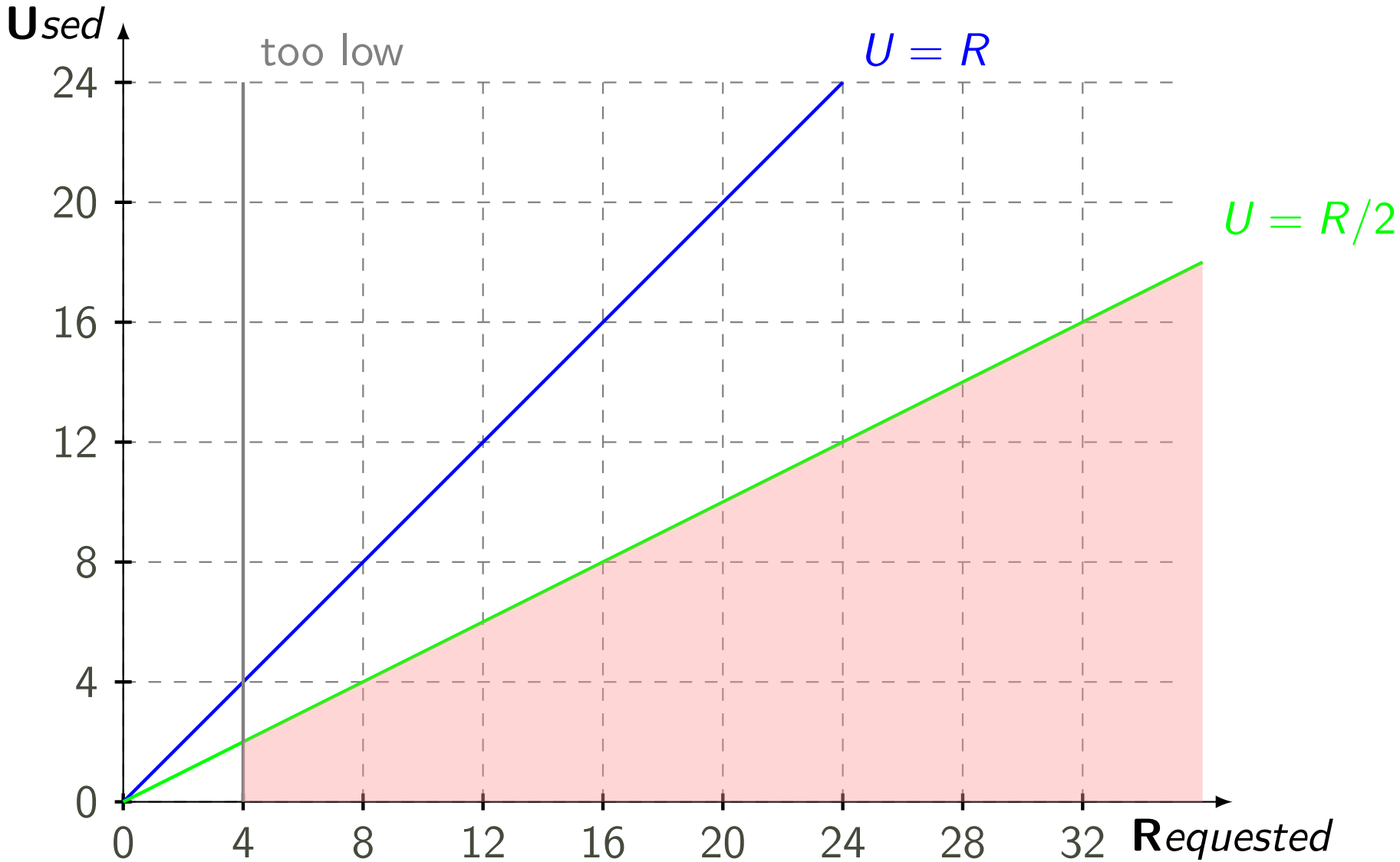
# Walltime example



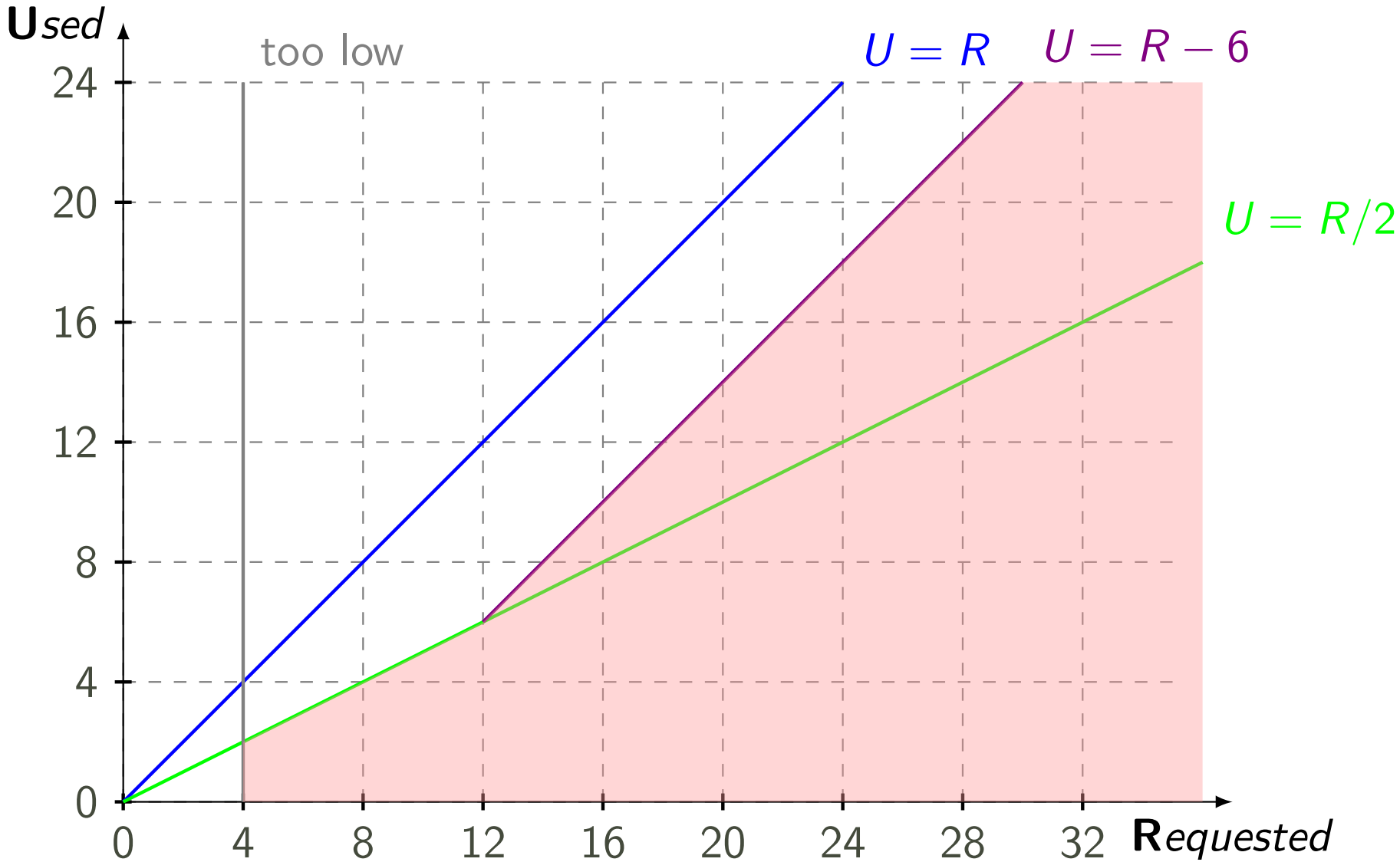
# Walltime example



# Walltime example



# Walltime example



# Rules and Attributes

**Attribute** is a property of the job with predefined values

$$A \in \{V_1, V_2, ..., V_n\}$$

For example, *used_walltime* attribute might be `LOW` or `NORM`

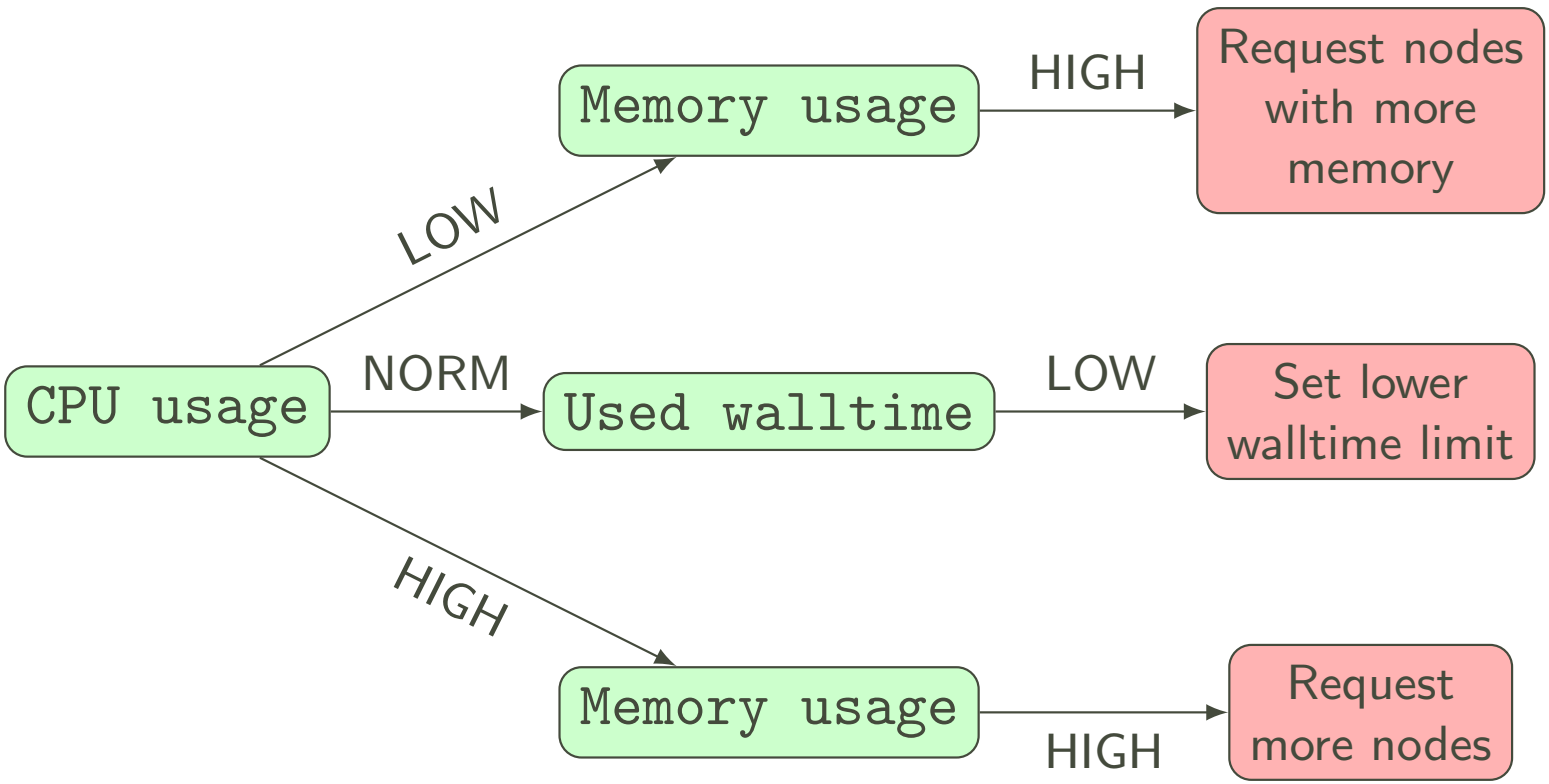
**Rule** is a set of attributes with specified values

$$R : A_1 = V_1 \ \& \ A_2 = V_2 \ \& \ ... \ \& \ A_k = V_k$$

For example, *fix_requested_walltime* rule might be formed by attributes like *used_walltime* = `LOW` & *cpu_usage* = `NORM`

# Decision trees

The set of the rules can be represented and visualized as a decision tree



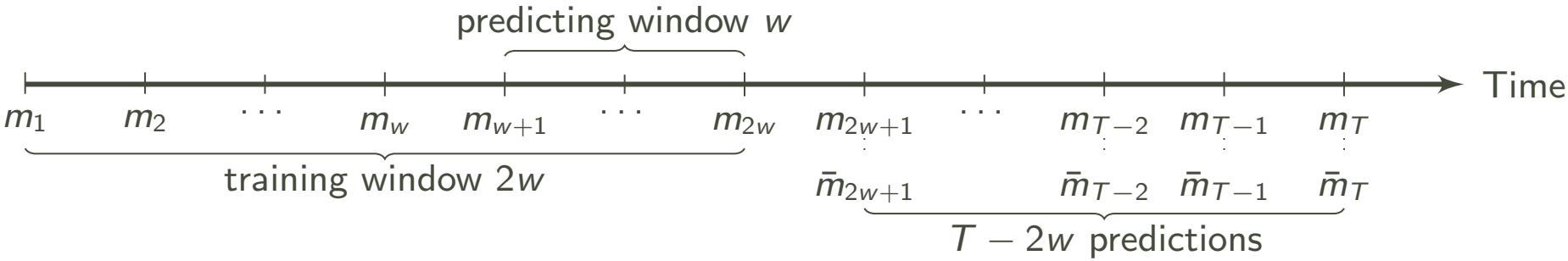
# Anomaly detection

# Anomaly detection

## General problem

Calculate anomaly score of a program using measurements during a runtime

# Anomaly detection implementation

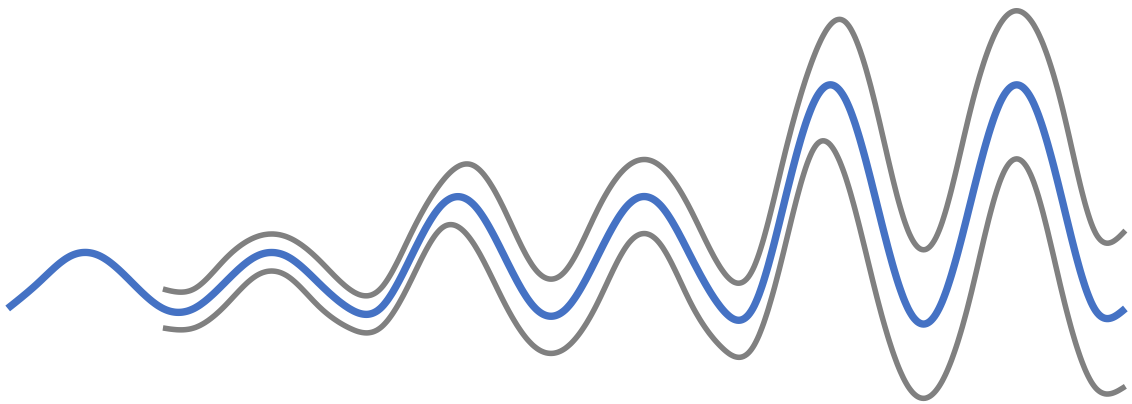



1. Neural Network is trained on previous measurements every time step
2. It makes prediction for the next time step
3. Prediction is compared to the observed value and anomaly score is calculated

# Anomaly Score

## Threshold

- Points with higher anomaly score than threshold considered to be anomalous
- Threshold dynamically calculated using standard deviation of previous points
- Adjustable with parameters (scaling and smoothing)



# Anomaly detection example

## Anomaly detection of GPU utilization with neural networks

# Anomaly detection challenges


- Such anomaly detection mechanism produces false positive results, especially when a program has various runtime phases
- Measurements usually have large amount of noise
- Measurements sometimes don't have patterns
- Proposed anomaly detection models were trained using measurements which were taken *every second*


Thank you!

Questions?

https://profit-hpc.de info@profit-hpc.de