# Walltime Prediction and its Impact on Job Scheduling Performance and Predictability

Dalibor Klusáček
*(CESNET, Czech Republic)*

Mehmet Soysal
*(Karlsruhe Intitute of Technology, Germany)*

**JSSPP 2020**

cesnet

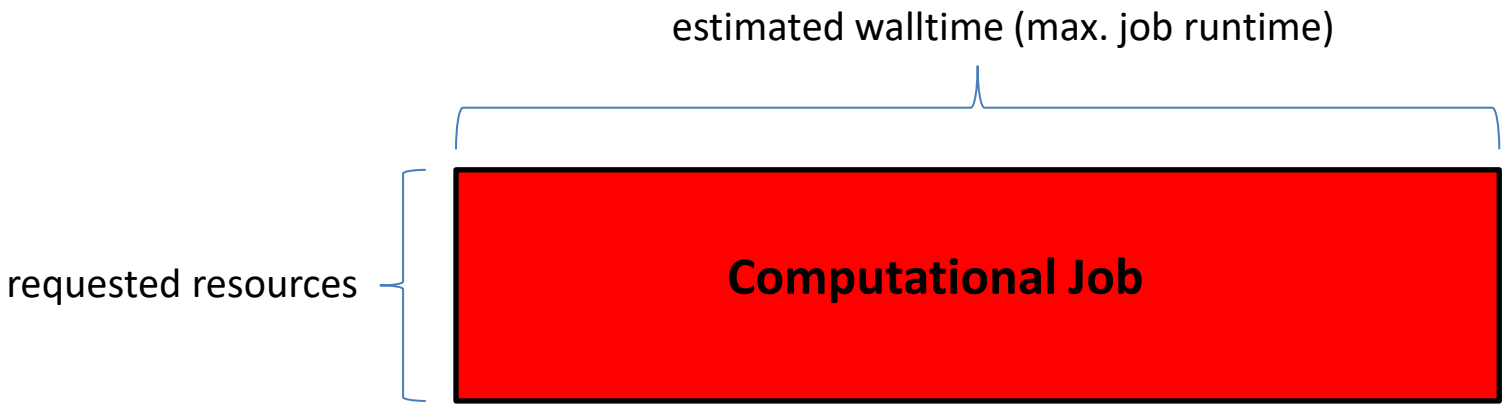
Karlsruher Institut für Technologie

# Talk Motivation

- Cesnet and KTH both operate large HPC infrastructures
- Both institutes consider to **automatically adjust (refine) user-provided job walltime estimates**
- **Is it worth trying?** Will it bring some benefits?
  - Better performance and/or predictability?
  - Are we able to use them for new features?

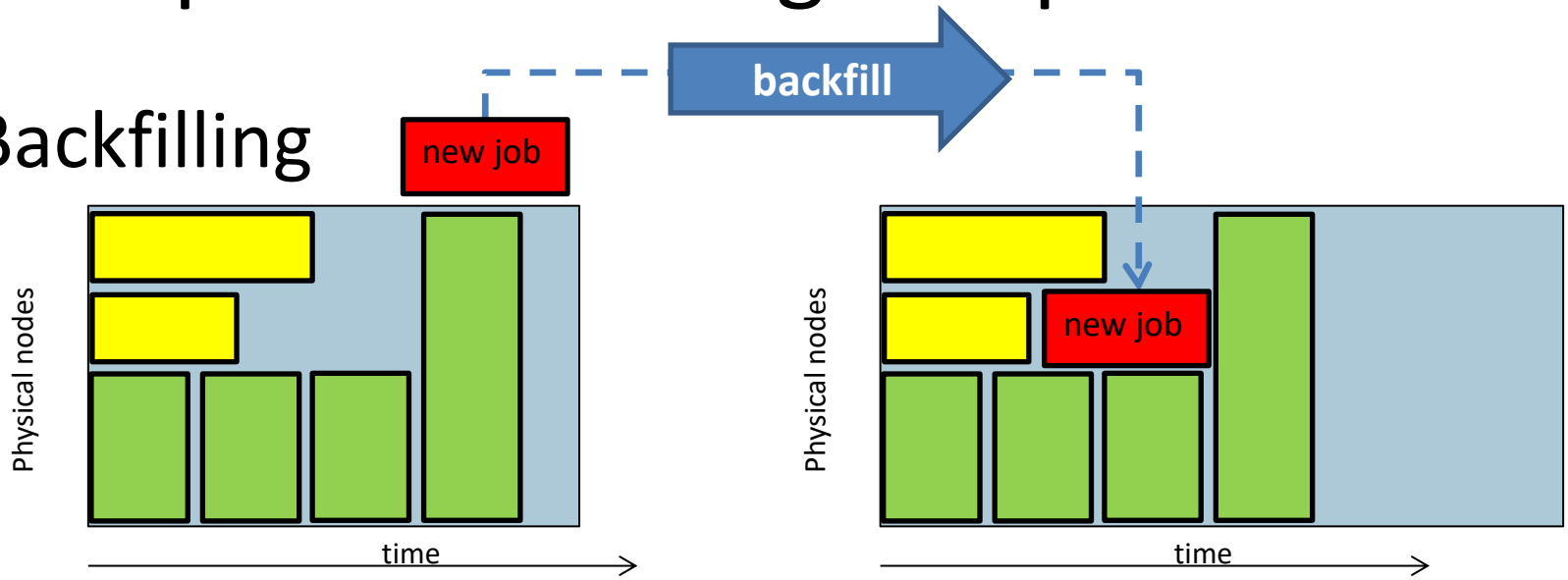
# Walltime estimates

- Batch job schedulers rely on user-provided walltime estimates
  - To know when to kill a job
  - To optimize the performance (backfilling)
  - To predict (when and where will a job start)
  - To stage data ahead of computation

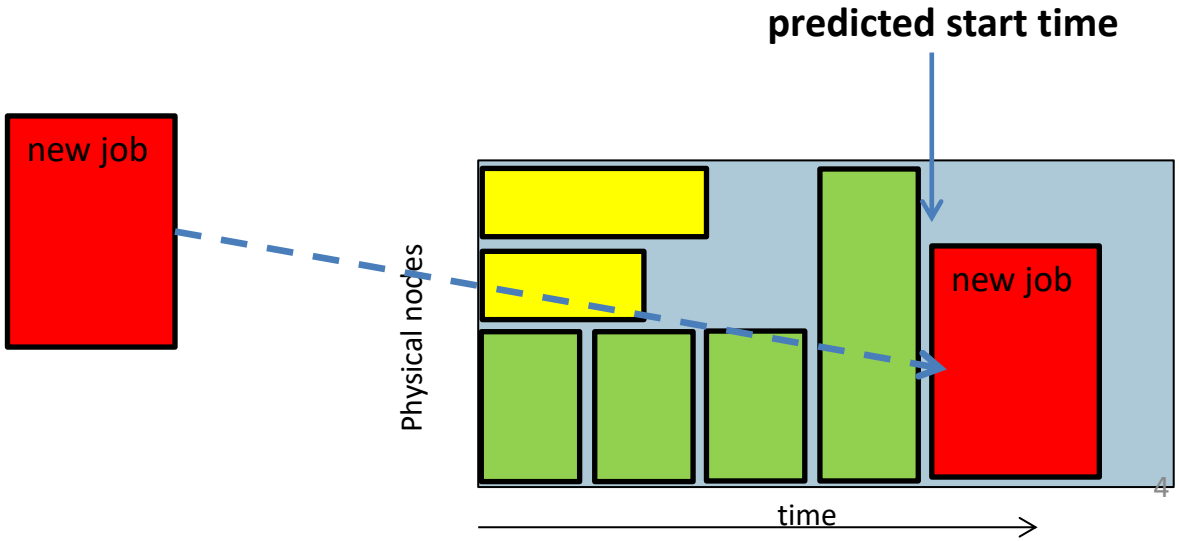


# Examples: backfilling and predictions

- Backfilling

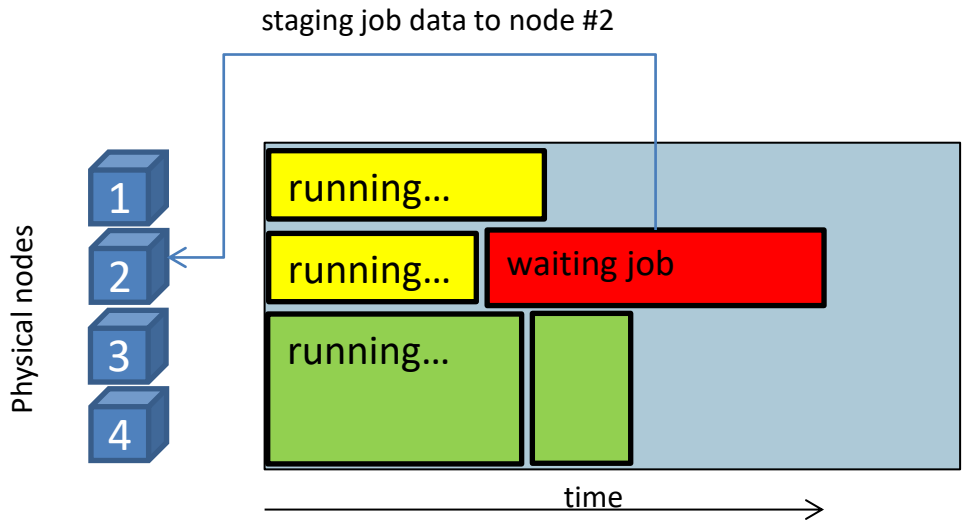



- Prediction

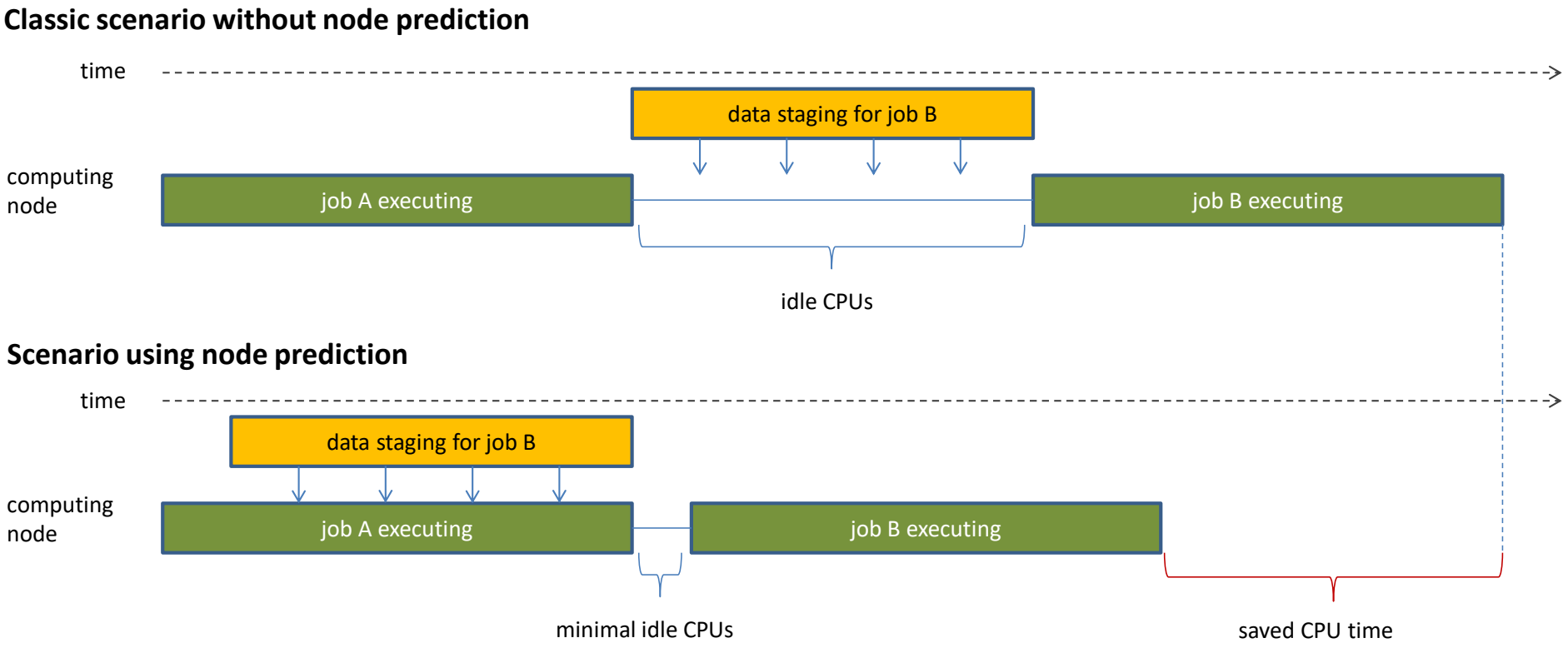


# Example: Node prediction and data staging

- Data staging ahead of computation

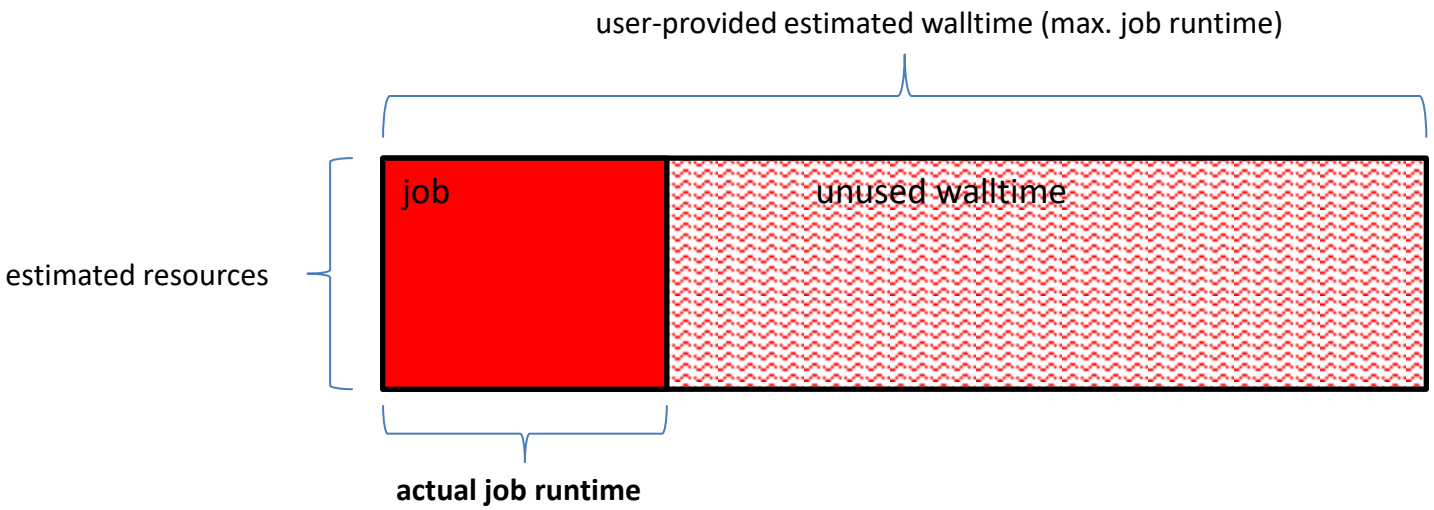


# Node prediction and data staging



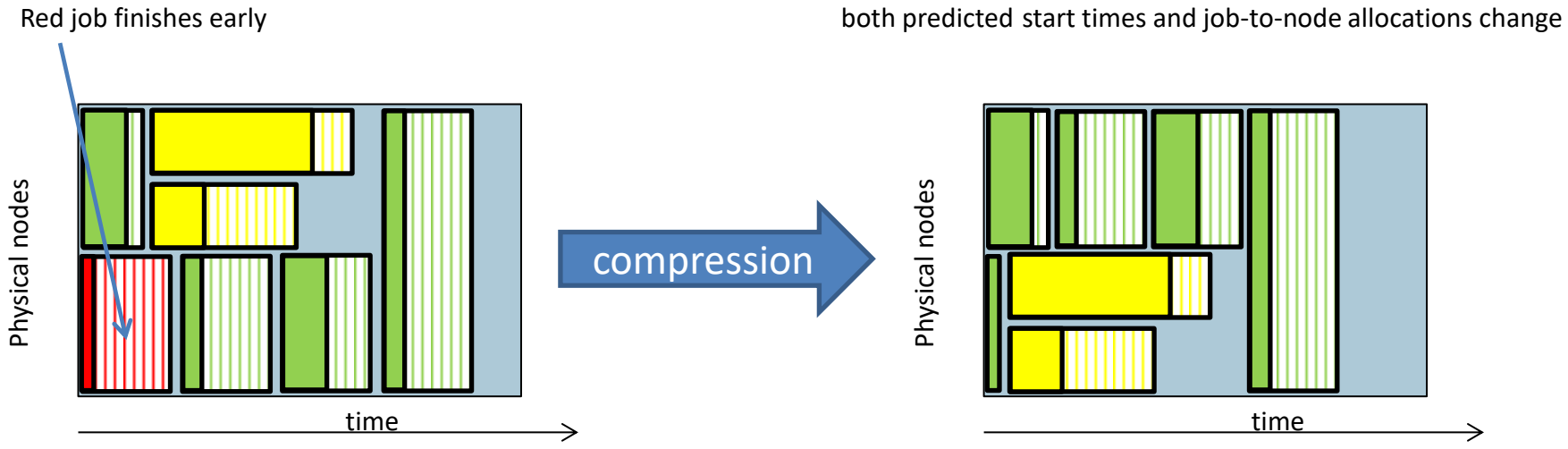
# (Inaccurate) walltime estimates

- Estimates are typically over-estimated and coarse grained
  - To prevent jobs from being killed due to exceeding their walltime



# Impact of inaccurate walltimes

- Schedule is getting “shorter” than expected
- Triggers **schedule compression**
  - invalidates predictions
  - shuffles job-to-node allocation



# Let’s improve the walltime estimates!

- **Use a walltime predictor**
  - A simple one, using user’s history
  - Adjusting user-provided walltime according to the recent user’s “overestimation ratio”

- **See if it helps!**
  - Compare performance
    - user-provided estimates vs. predictor
  - Simulations performed using 4 workloads
  - KIT FH1, KIT FH2, CTC, SDSC

# What we observed

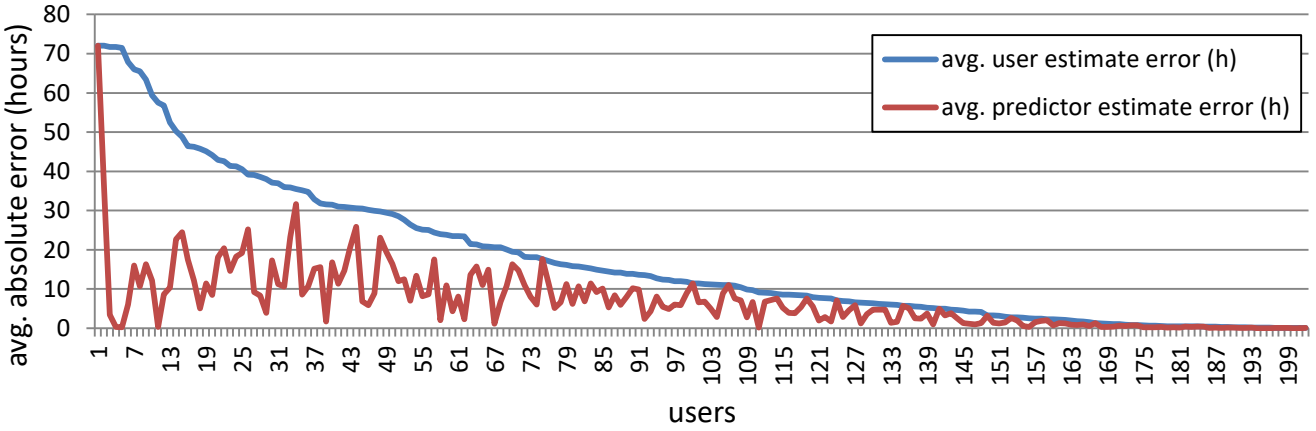
- Accuracy of estimates (users vs. predictor)
- Backfilling ratio
- Wait time distribution
- Accuracy of predicted job wait time
- Accuracy of node allocation predictions

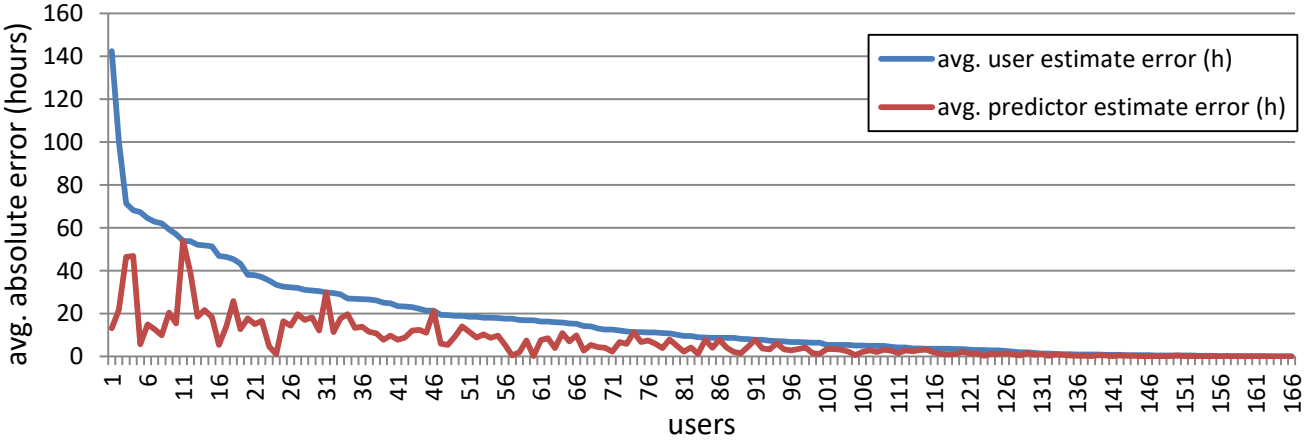
# Quick summary

- Accuracy of estimates (users vs. predictor) ☺
- Backfilling ratio ☺
- Wait time distribution 😐
- Accuracy of predicted job wait time ☺
- Accuracy of node allocation predictions ☹

# Accuracy of user estimates vs. predictor

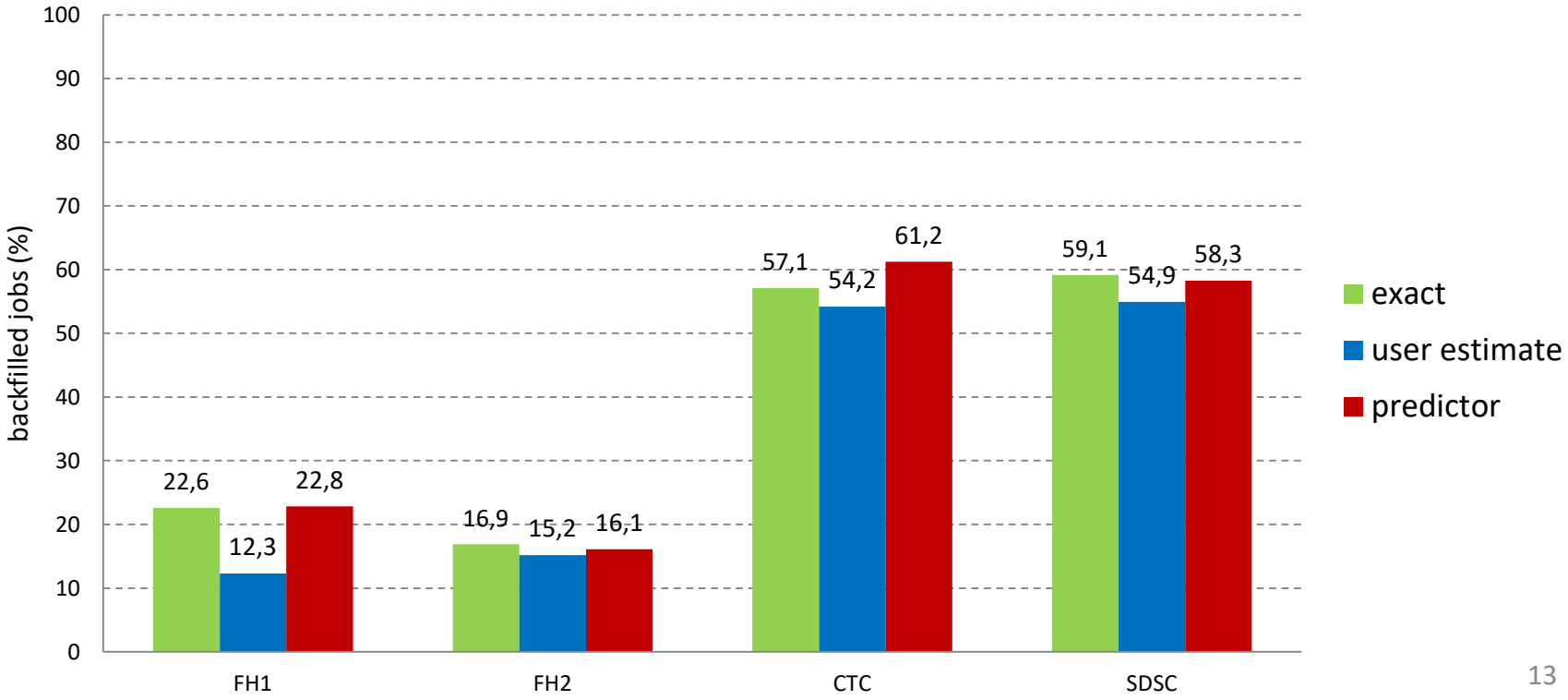
- Avg. absolute error of walltime estim. (per user)



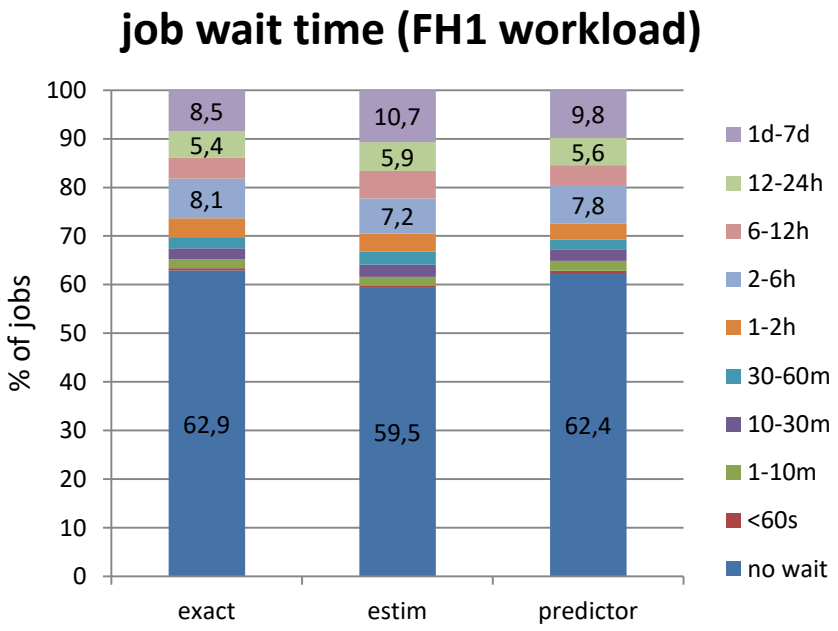
# Backfilled jobs

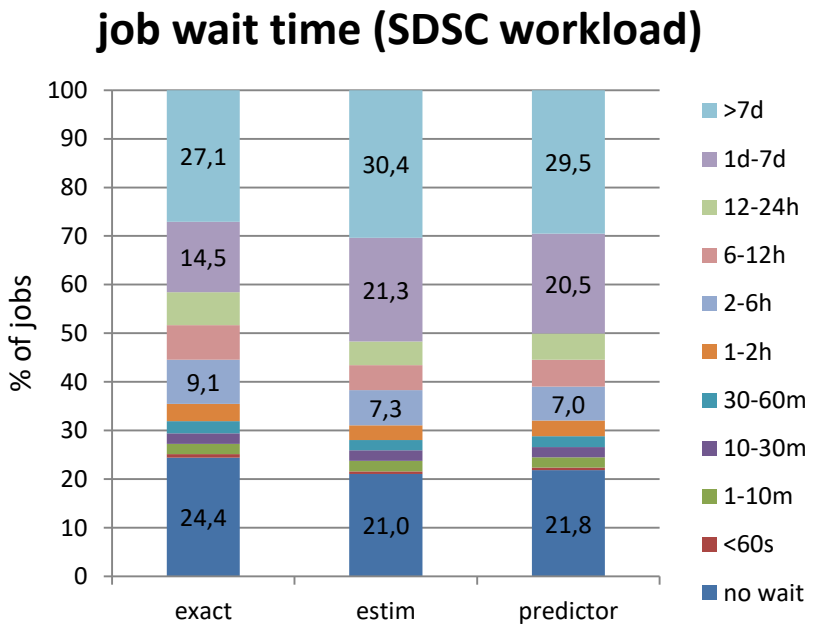
- Percentage of backfilled jobs increases
  - No dramatic effect
    - Better estimates => smaller holes in the schedule



# Wait time

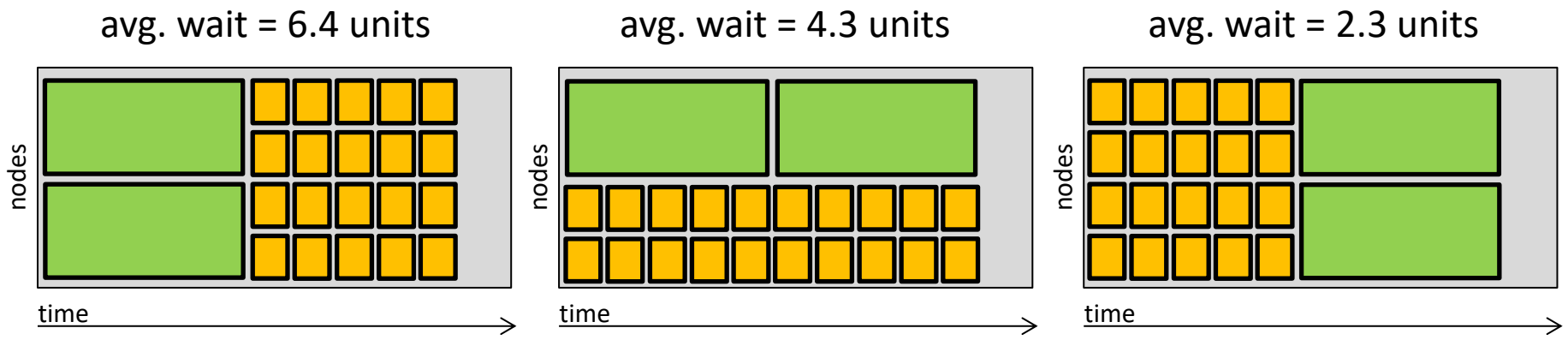
- Avg. wait time was improved (few extremes)
  - Wait time distribution shows only slight improvements (if any)



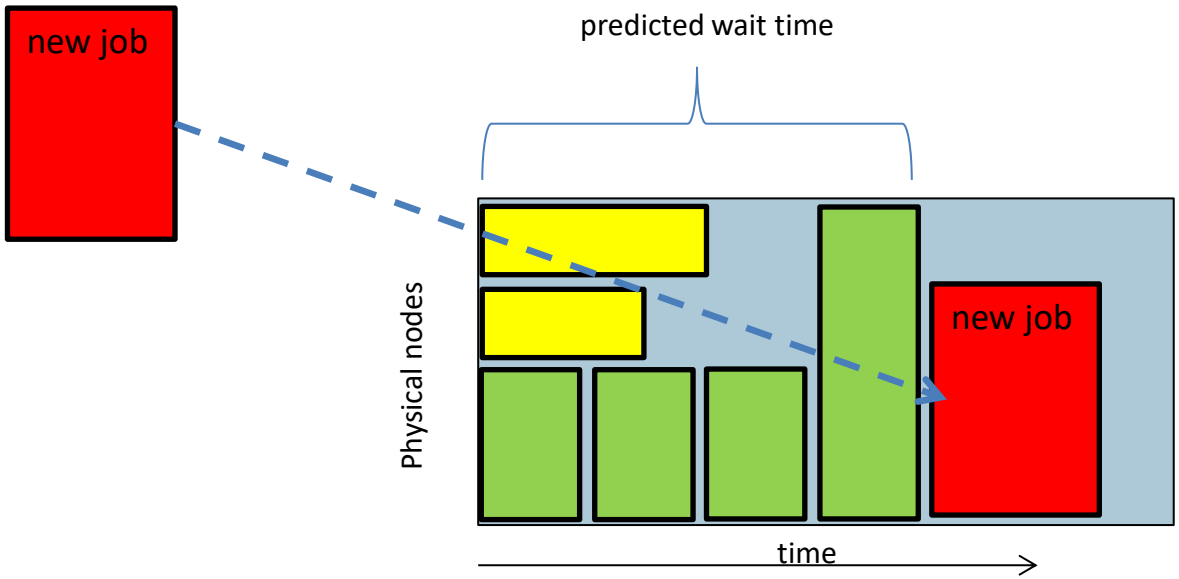
# Wait time is “tricky”

- Depends on many factors
- Can be influenced by job ordering
  - Shortest Job First tendency of backfilling
  - Improving estimates => making holes smaller too

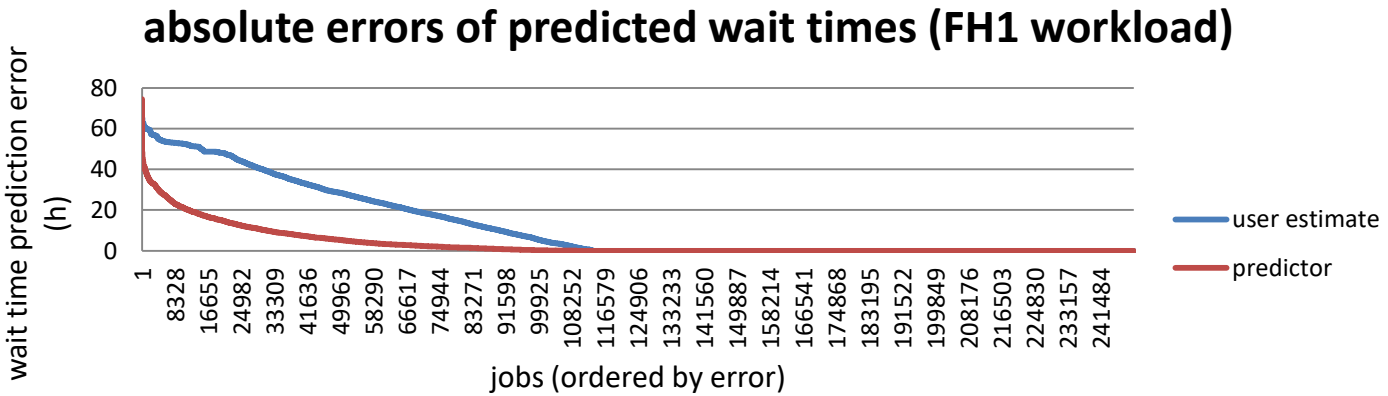


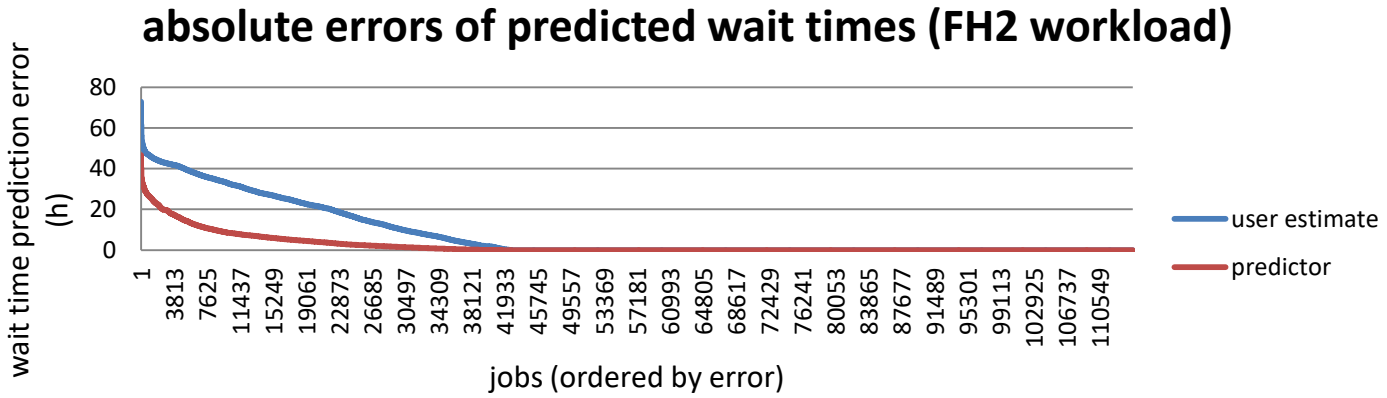
# Accuracy of wait time predictions

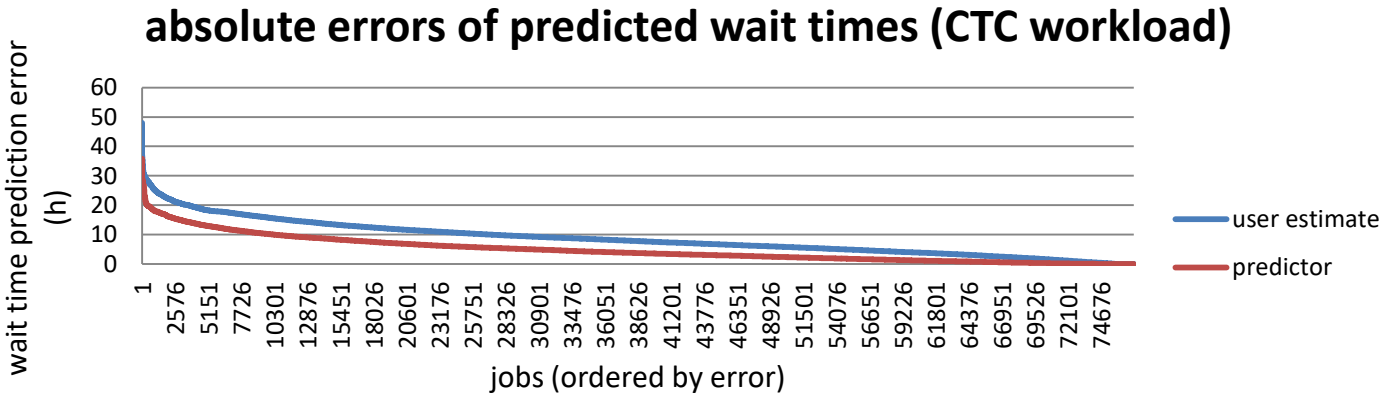
- Predicted wait time
  - **Measured upon job arrival** (initial prediction)
  - Further improved as schedule is compressed over the time due to early finishing jobs



# Wait time prediction errors (hours)

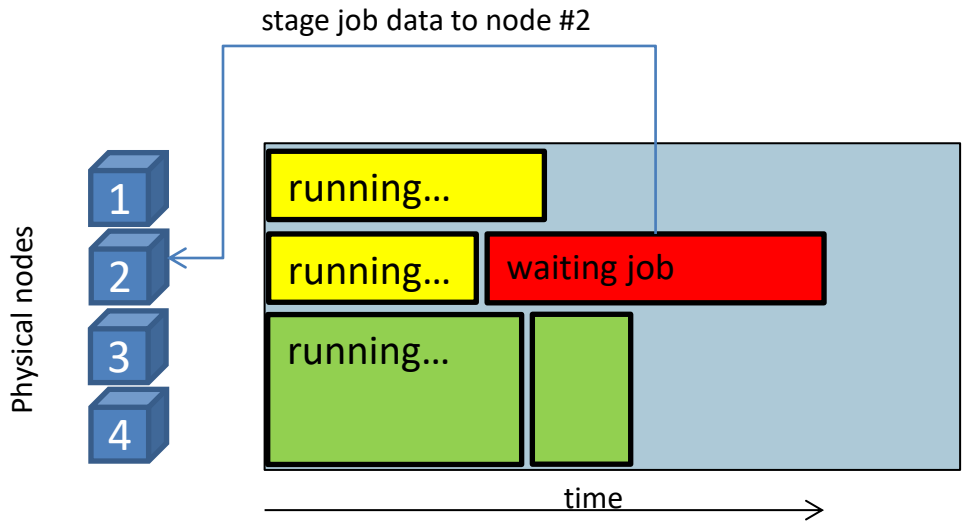


# Node allocation predictions

- **Can we know ahead of job start where it will be executed?**
- Stage in data in advance (saving CPU cycles)

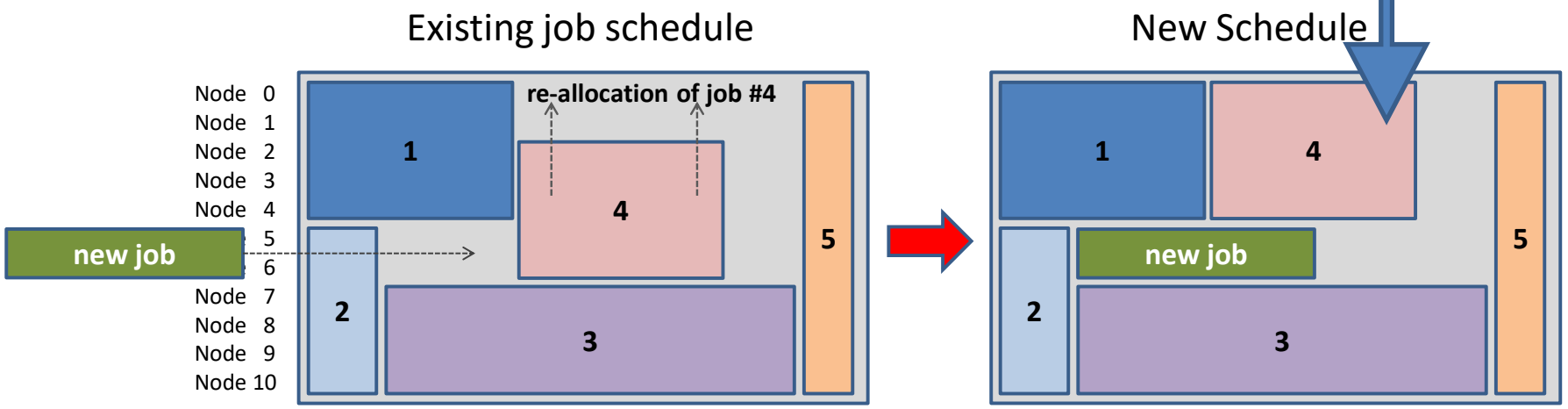


# Node allocation predictions

- **Can we know ahead of job start where it will be executed?**
  - With exact runtimes, sure
  - With user estimates and predictor?
  - **Not really ☹**
- **Why?**
  - Simply put, we were lazy to develop a brand new scheduling algorithm
  - Fixing existing ones didn't work…

# Why do we fail?

- Several reasons
  - Backfilling shuffles jobs around
  - Schedule compression does the same
  - **Job-to-node allocations change a lot**



# Can we improve?

- Backfilling → **FCFS**
  - FCFS helps, but… **we pay the cost**
  - Poorer utilization
  - Much worse wait time
  - **We still need to compress the schedule**

- That goes against our primary motivation to reduce idle CPU time ☹

# Can we improve?

- Schedule compression → **job-to-node pinning**
  - Pin jobs that are about to start soon to their planned nodes
  - Same drawbacks (utilization & wait time)

- That goes against our primary motivation to reduce idle CPU time ☹

# Node allocation prediction example

- **How many jobs know at least 1 minute ahead of their start time what is their target node?**
- Assuming FCFS technique applied[1]:
  - With precise runtimes: 88%
  - With user estimates: 12%
  - Using predictor: 17%
  - Using job-to-node pinning: 58%
- So far, quite a nice improvement…

# Node allocation prediction - cost

- We managed to go from 12% to 58% ☺
- **But what is the cost (in wait time)?**

- With our solution (FCFS+node pinning)
  - **18% jobs wait longer than 24h** ☹

- With standard backfilling without pinning:
  - **only 10% jobs wait longer than 24h** ☺

# Conclusion

- Even simple predictor increases accuracy of estimates
- It also improves backfilling opportunities
- Wait time is only slightly reduced
- Wait time (start time) predictions are improved significantly
- Node predictions are hard with standard algorithms
  - either precise or efficient schedule, but not both

# Future work

- Node predictions are hard with standard algorithms
  - either precise or efficient schedule, but not both
- Use **better scheduling** algorithms
  - Jobs do not overlap
  - So when one finishes earlier, no gap appers
- **Not all jobs really need this treatment**
  - To stage in advance / to be pinned

# Thank you!

klusacek@cesnet.cz

cesnet

Karlsruher Institut für Technologie