SAND2022-15091C

UCF | AREA 67

UNIVERSITY OF CENTRAL FLORIDA

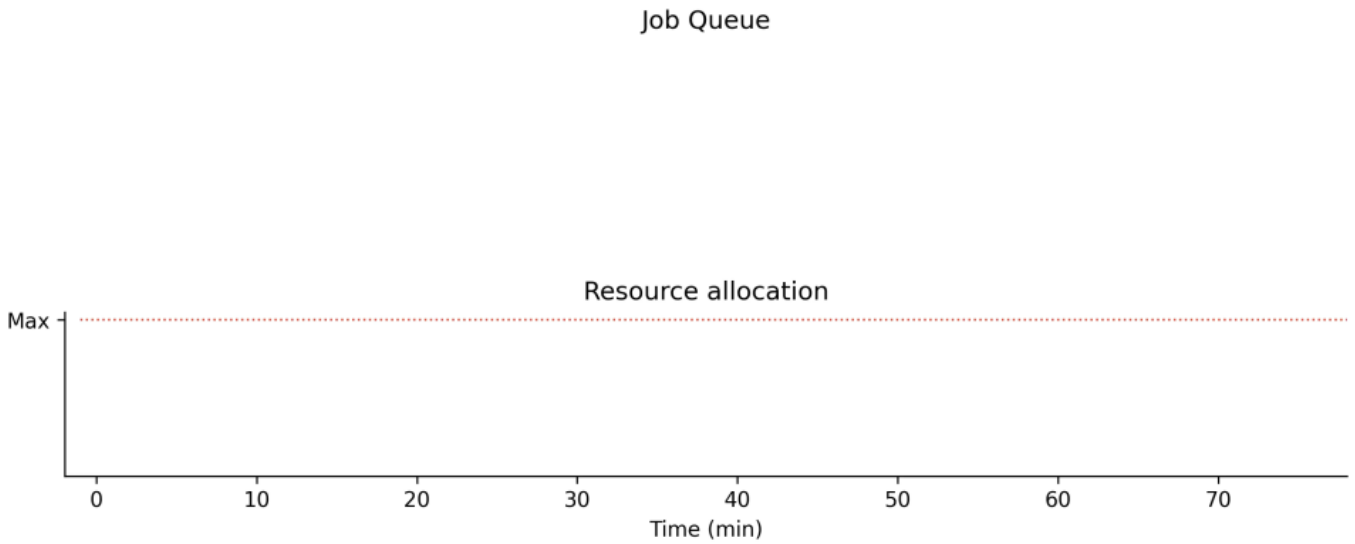
Sandia National Laboratories

Metrics for Packing Efficiency and Fairness of HPC Cluster Batch Job Scheduling

Alexander V. Goponenko (UCF), Kenneth Lamar (UCF), Christina Peterson (UCF),
Benjamin Allan (SNL), Jim M. Brandt (SNL), and Damian Dechev (UCF)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Job scheduling on HPC cluster



- Users submit jobs and specify jobs' resource requirements
  - Essential parameters:
    - Number of nodes
    - Time limit
- Scheduling algorithms determine the order of starting jobs during shortage of resources
- "Online" scheduling problem
  - NP-hard

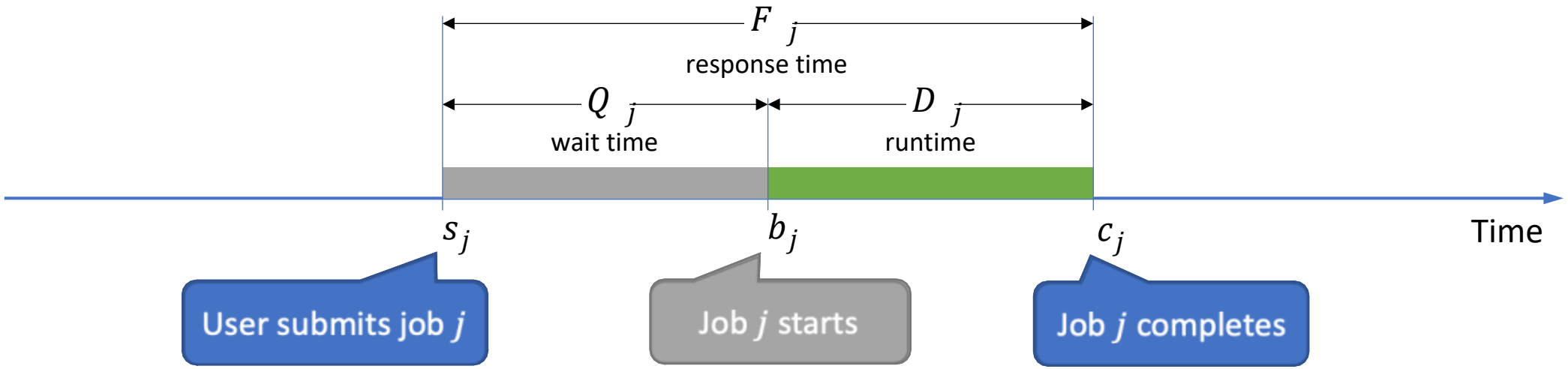
Goals of HPC cluster scheduling

Goal 1: Efficiency
- job packing efficiency

Goal 2: Complying with priorities
- priority policy
- fairness: job that waited longer should have higher priority

Goal 3: System responsiveness as perceived by users
- response time

Lifetime of a job



Assumptions

- One resource only
  - e.g. nodes
- No job runtime depends on how jobs are scheduled
  - i.e. no I/O
- No job arrival time depends on how jobs are scheduled

Popular HPC Schedule Metrics

- **Average response time**
  - **Used for:**
    - **Goal 1**: Efficiency
    - **Goal 3**: System responsiveness as perceived by users
- **Average slowdown**
  - **Used for:**
    - **Goal 3**: System responsiveness as perceived by users
- **Utilization**
  - **Used for:**
    - **Goal 1**: Efficiency

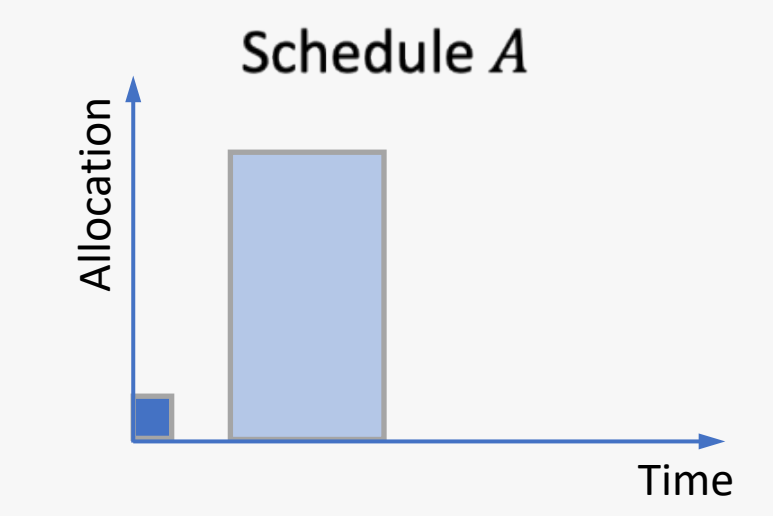
Average Response (Flow) Time (AF)

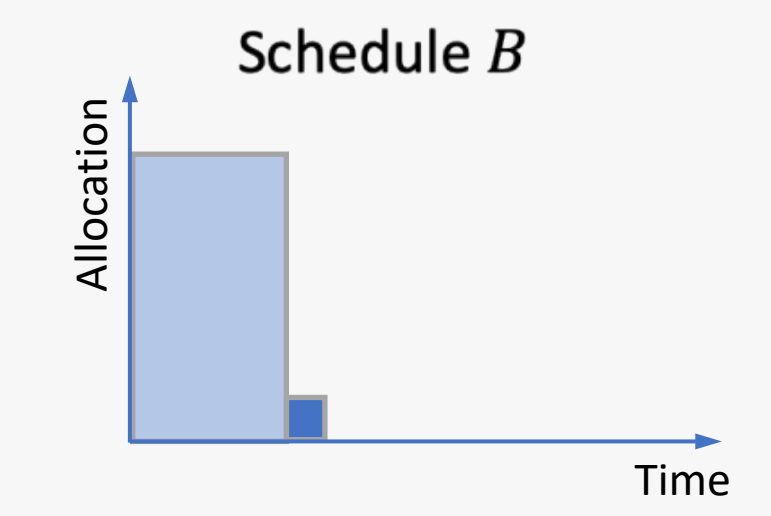
$$AF = \frac{1}{n} \sum_{1 \le j \le n} F_j$$

number of jobs

- Equal weights for all jobs
- Improves when starting smaller jobs earlier and delaying larger jobs

$$AF(A) = AF(B)$$





Average Slowdown

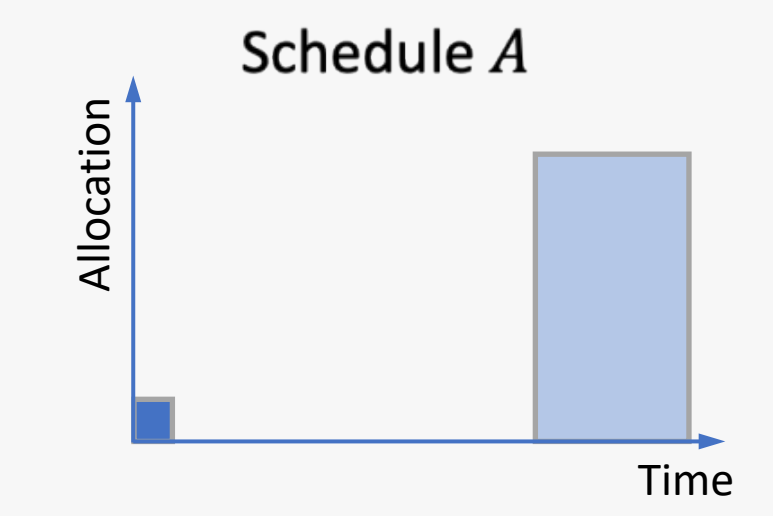
$$Sld = \frac{1}{n} \sum_{1 \le j \le n} \frac{F_j}{D_j}$$

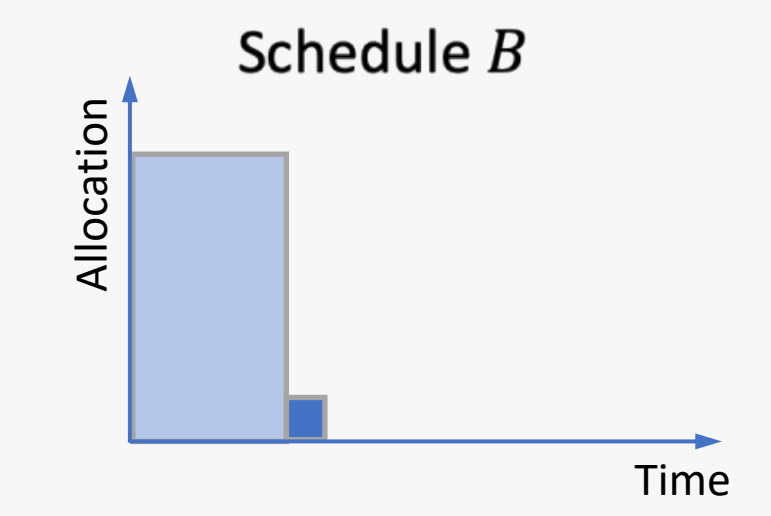
response time

runtime

- Dramatically improves when starting smaller jobs earlier
- A job with infinitely small runtime makes an infinitely large contribution

$$Sld(A) < Sld(B)$$





Average Bounded Slowdown ($BSLD$)

$$BSLD = \frac{1}{n} \sum_{1 \le j \le n} \max\left(1, \frac{F_j}{\max(D_j, \kappa)}\right)$$

parameter (constant)

- A crude patch to alleviate dramatic contribution of small jobs
  - Same weights (as in AF) for jobs satisfying $D_j < \kappa < F_j$
  - No contribution from jobs satisfying $F_j < \kappa$

- In practice, κ is usually small (e.g. 10 s)
- Still, $BSLD$ dramatically improves when starting smaller jobs earlier

$Utilization$ is determined by maximum completion time

$$Utilization = \frac{\sum D_j r_j}{R \times \left(\max_{1 \le j \le n} (c_j) - \min_{1 \le j \le n} (s_j) \right)}$$

resource requirement

total amount of resource

completion time

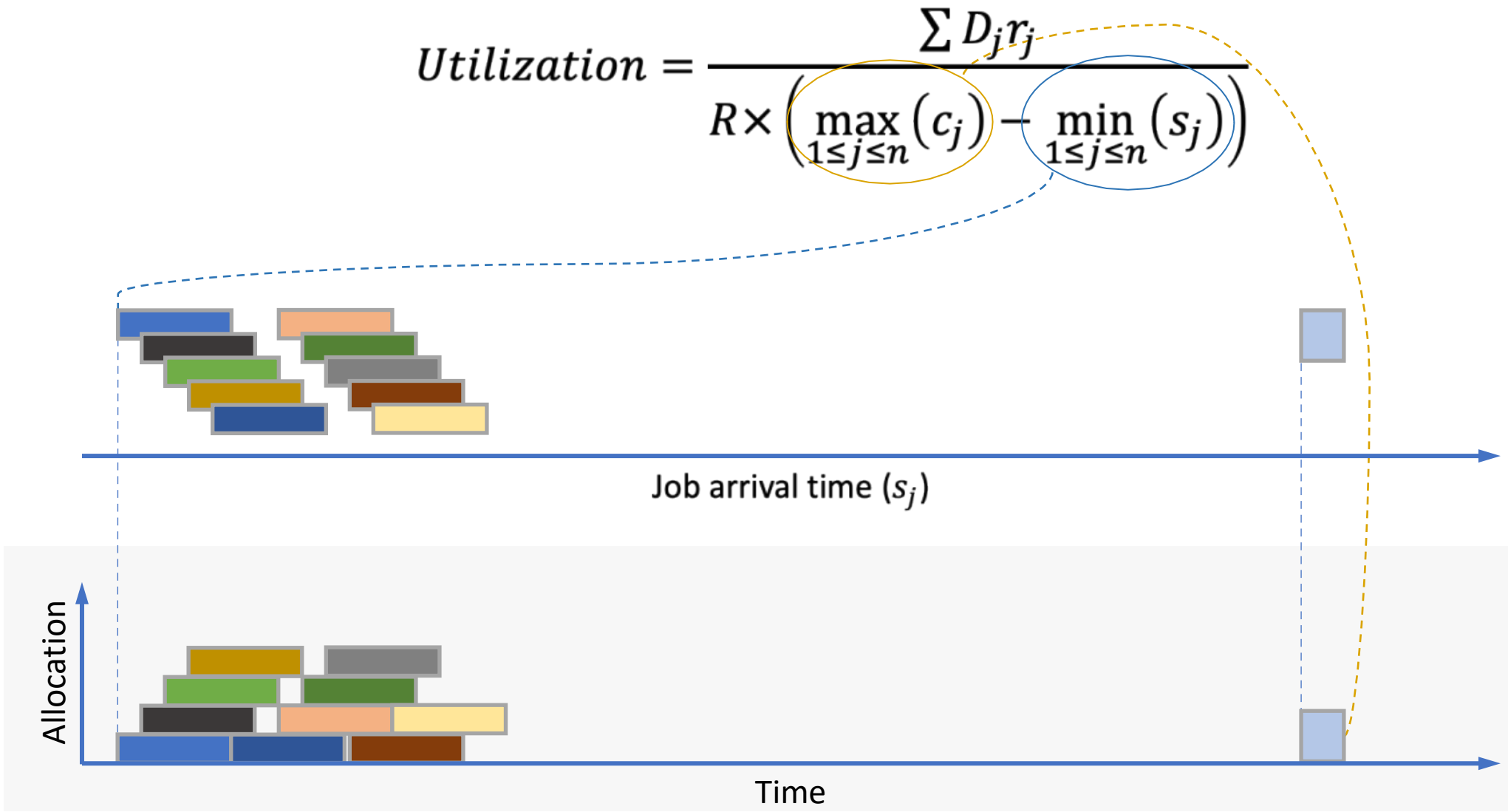
submit time

- "Offline scheduling" metric
- Not suitable for online scheduling

UCF

$Utilization$ is determined by maximum completion time

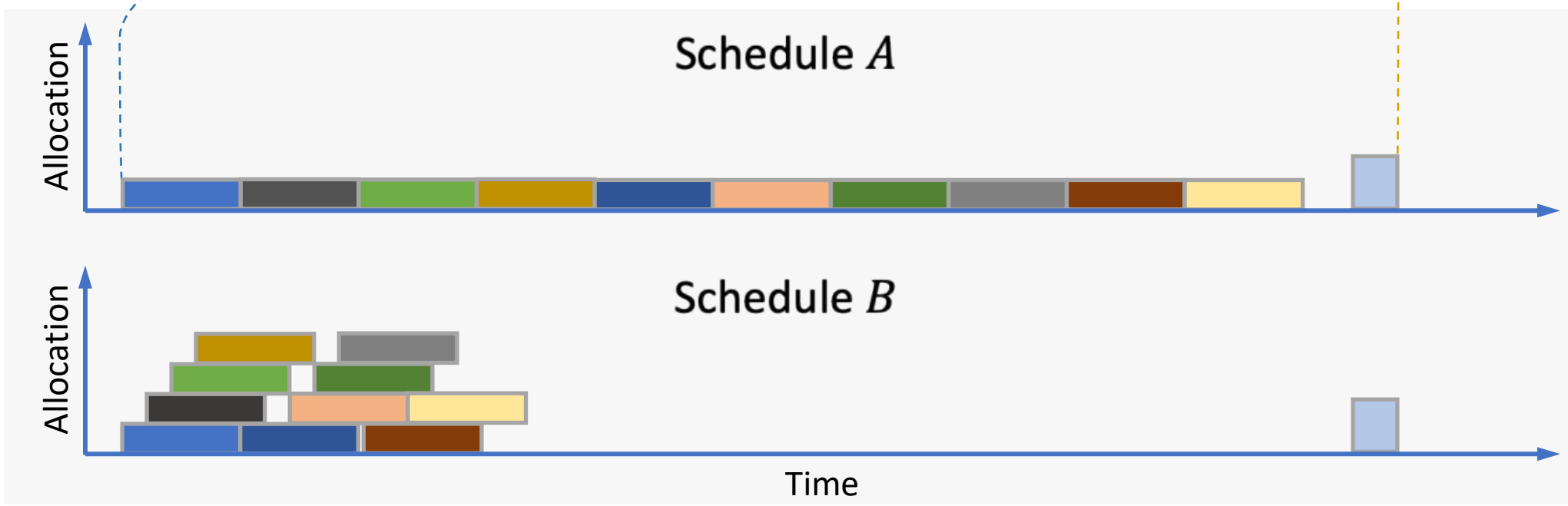
$$Utilization = \frac{\sum D_j r_j}{R \times \left(\max_{1 \le j \le n} (c_j) - \min_{1 \le j \le n} (s_j) \right)}$$



$Utilization$ is determined by maximum completion time

$$Utilization = \frac{\sum D_j r_j}{R \times \left(\max_{1 \le j \le n} (c_j) - \min_{1 \le j \le n} (s_j) \right)}$$

$$Utilization(A) = Utilization(B)$$



Metric coverage of scheduling goals

Goal 1: Efficiency
- Not covered by common metrics

Goal 2: Fairness and complying with priorities
- Not covered by common metrics

Goal 3: System responsiveness as perceived by users
- AF and $BSLD$

Outline

Metric for packing efficiency

Metric for fairness and packing efficiency

Example 1: Scheduling algorithms

Example 2: Runtime predictions

Metric for Packing Efficiency

Desired properties

- Easy to calculate
- Reacts to even slightest improvement of the packing
- Improves (reduces) if a job starts earlier
- Doesn’t change if overall resource allocation profile is unchanged
  - Can’t be fooled by starting smallest jobs first

UCF

Area-Weighted Average Response Time

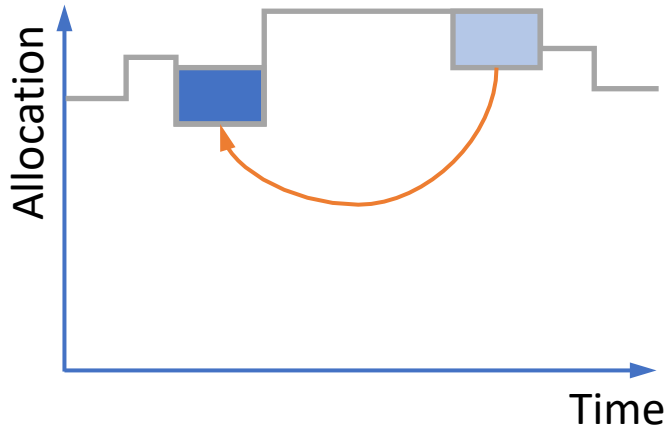
$$AWF = \frac{\sum r_j D_j F_j}{\sum r_j D_j}$$

resource requirement

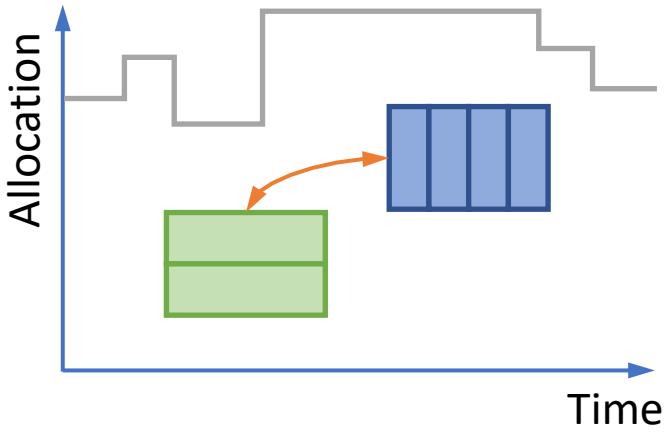
"squash" area

- Directly correlates to "job packing efficiency"

Moving a job to an earlier time reduces AWF

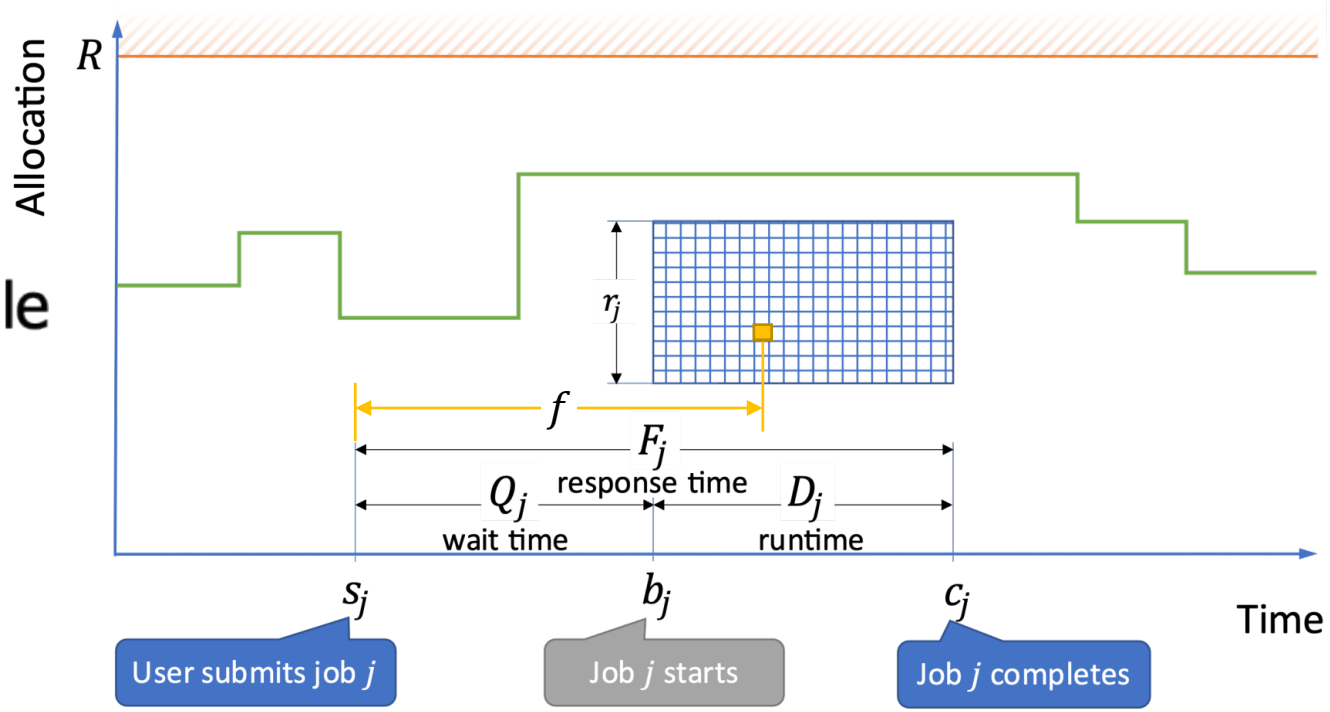


if resource utilization is not affected, reordering jobs doesn't change AWF



Metric for Fairness and Efficiency

- Desired properties
  - Improves (reduces) if a job starts earlier
  - If overall resource allocation profile is unchanged, reordering jobs-
    - doesn't change the metric if all jobs have same priority
    - reduced the metric if jobs with higher priority move to earlier time



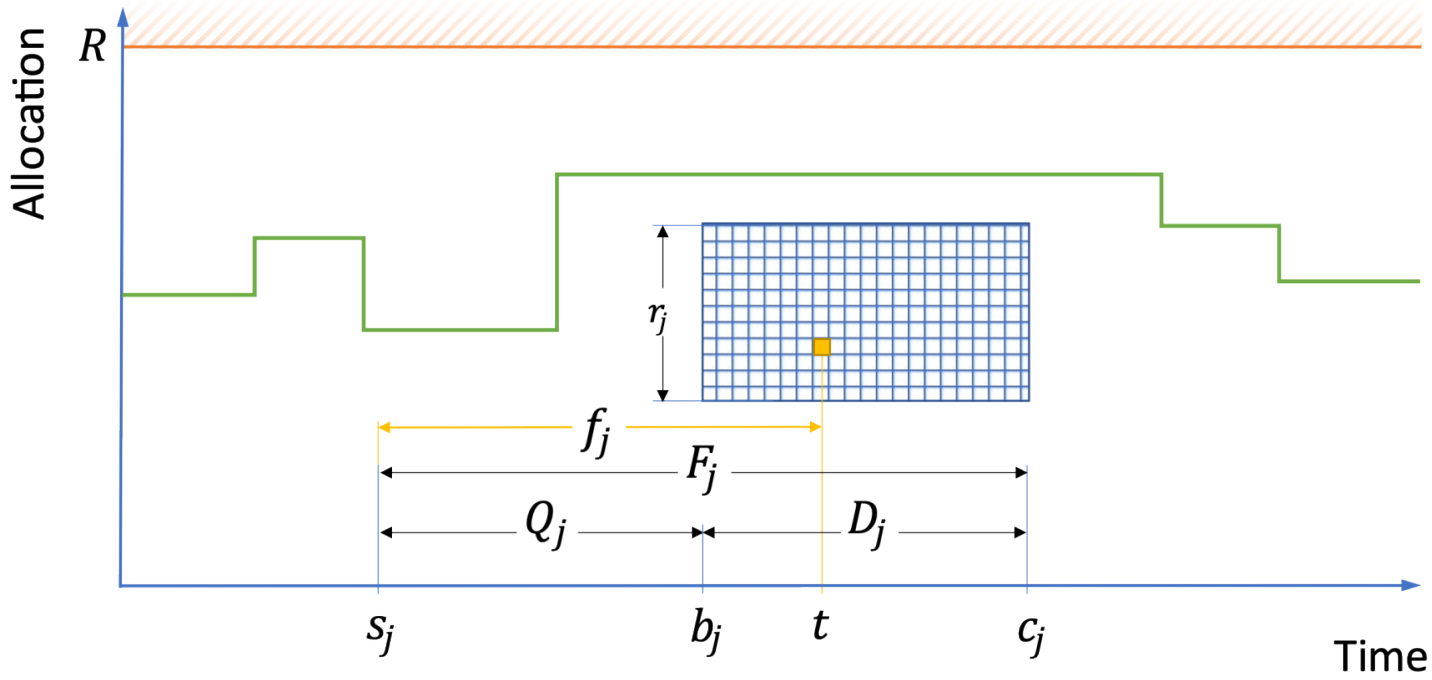
- Course of action
  - Treat each job as a set of small units of computation

L^{α}-Priority Weighted Specific Time

- Jobs are prioritized according to wait time

$$f_j(t) = t - s_j$$

$$p_j(t) = \left(t - s_j\right)^{\alpha}$$



$$P^{\alpha}SF = \frac{\alpha + 1}{\alpha + 2} \frac{\sum_j r_j \left(F_j^{\alpha+2} - Q_j^{\alpha+2}\right)}{\sum_j r_j \left(F_j^{\alpha+1} - Q_j^{\alpha+1}\right)}$$

Example 1: Scheduling Algorithms

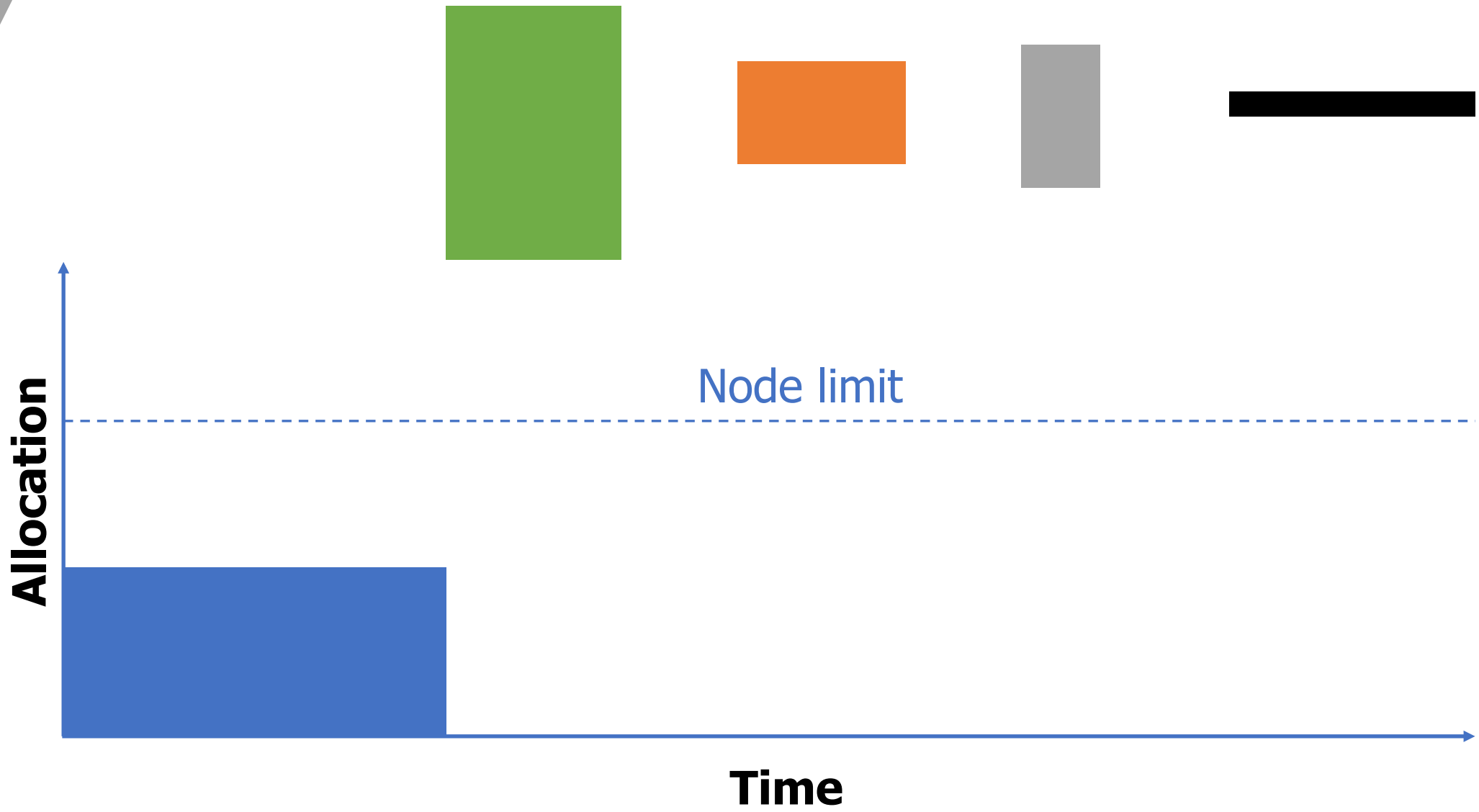
Motivation

EASY–Shortest-Job-Backfill-First algorithm is claimed to outperform normal EASY-Backfill because it improves $BSLD$.

Plan

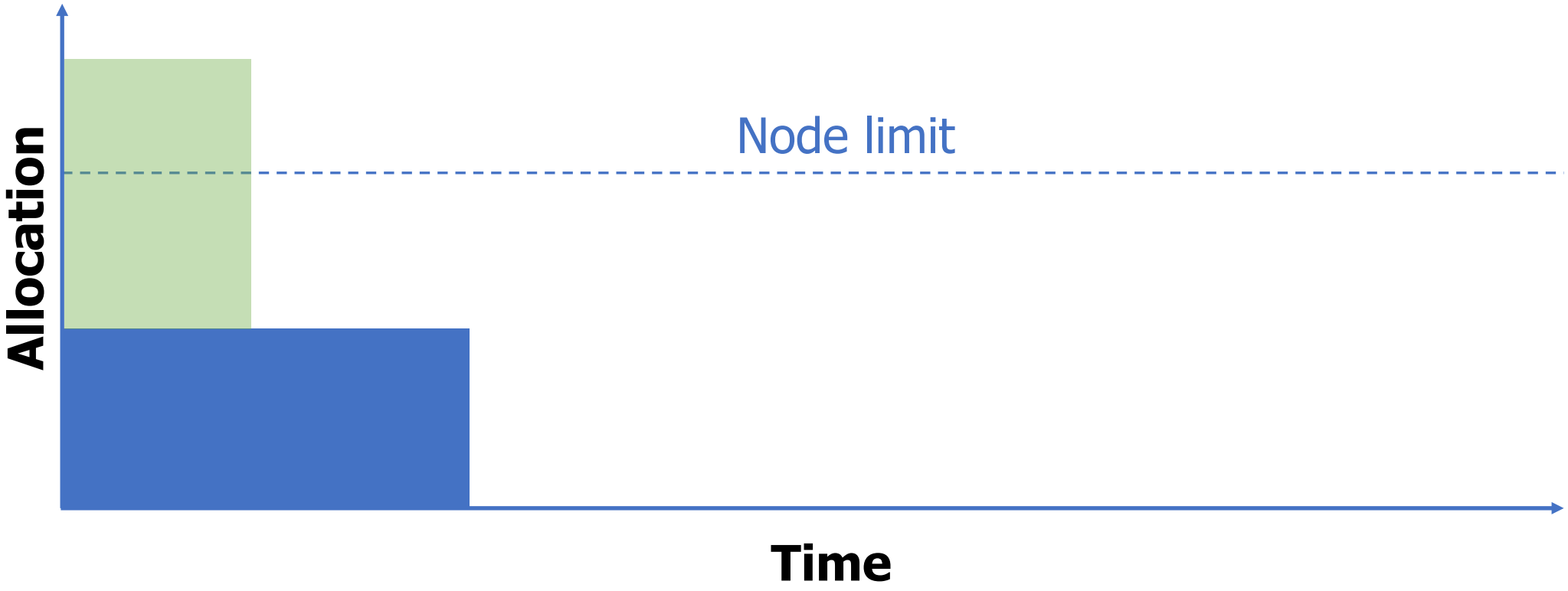
- Analyze this claim using $BSLD$, AF, AWF, and $P^{\alpha}SF$.
- Simulate scheduling of several real workload traces.
- Also analyze other basic variants of scheduling algorithms
  - "Just Backfill", "EASY", "Aggressive"
  - Queue orders: "FCFS", "SJF", "SAF", "LAF"

Just backfill (*JustBF*)



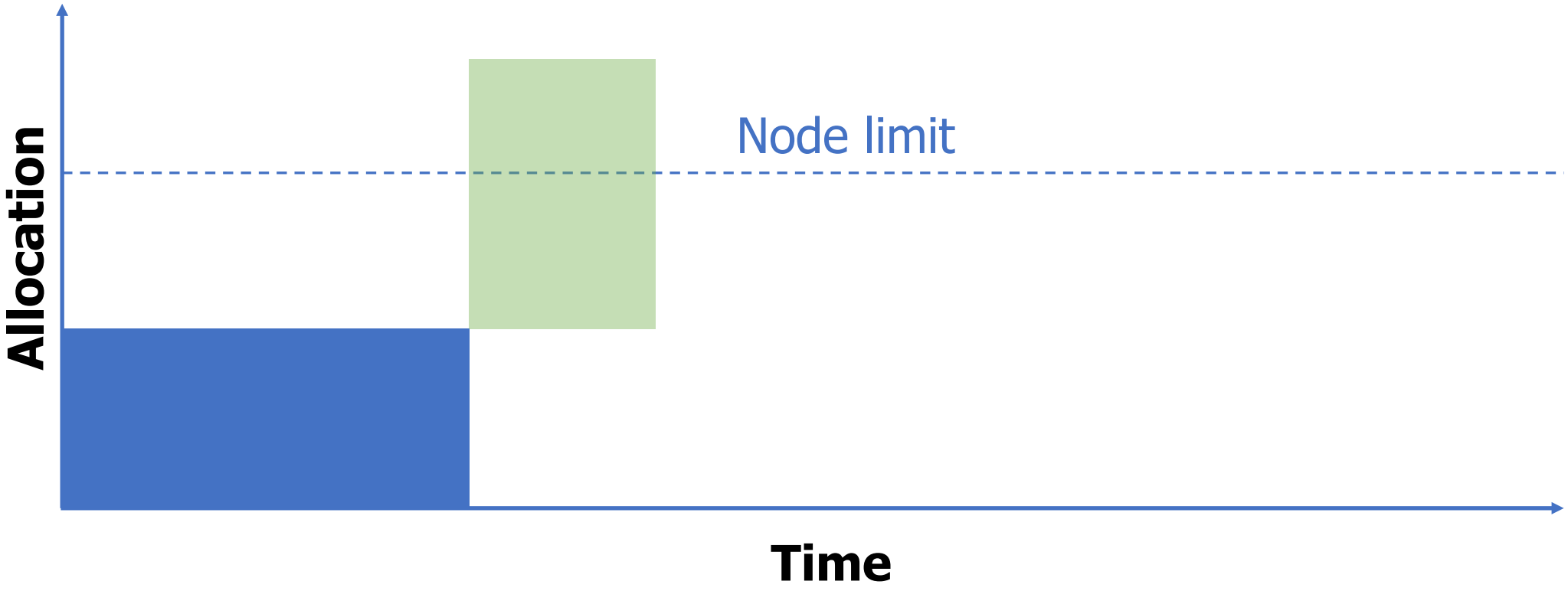
Just backfill (*JustBF*)

- If a job cannot run, reserve resources to prevent any further delays



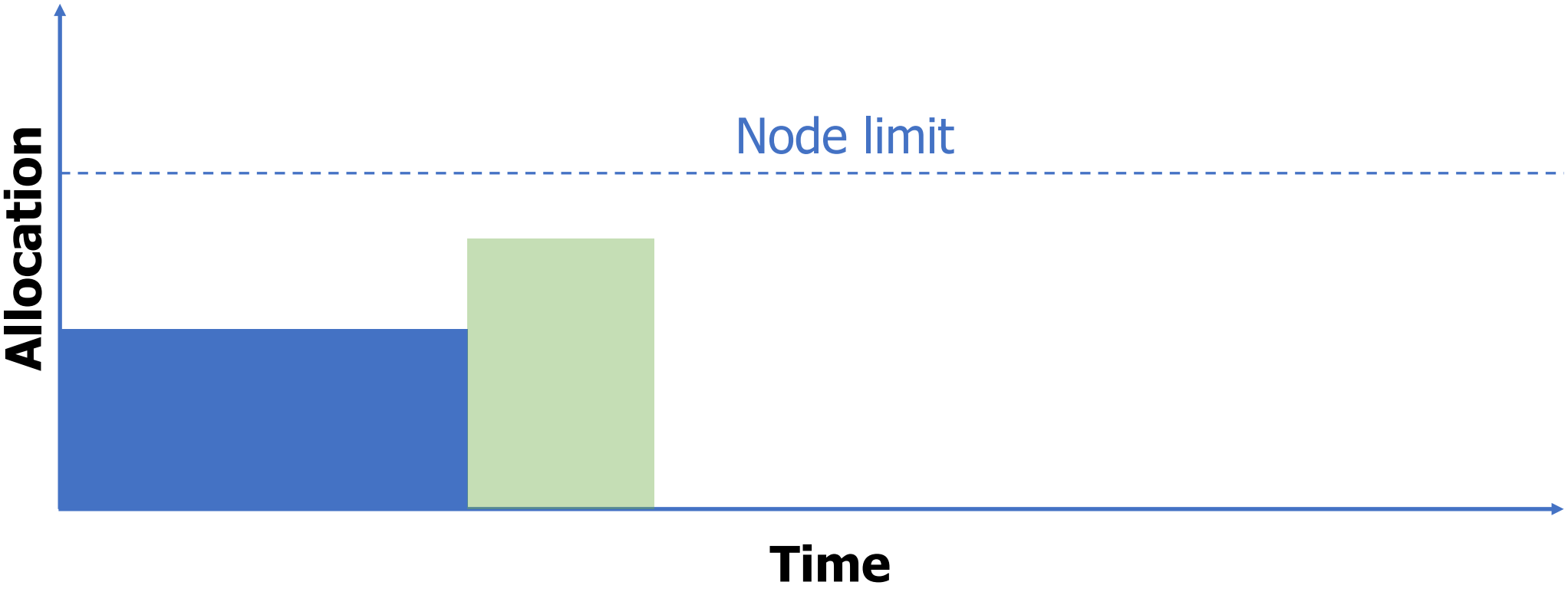
Just backfill (*JustBF*)

- If a job cannot run, reserve resources to prevent any further delays



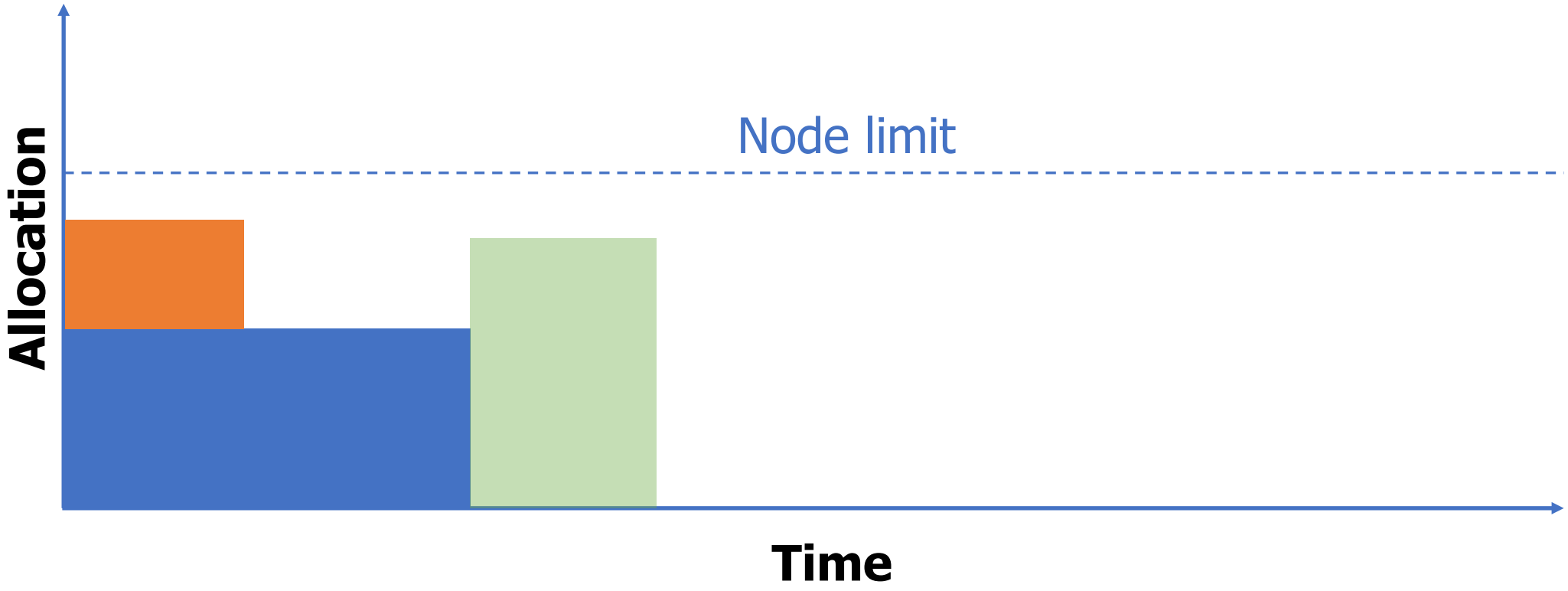
Just backfill (*JustBF*)

- If a job cannot run, reserve resources to prevent any further delays



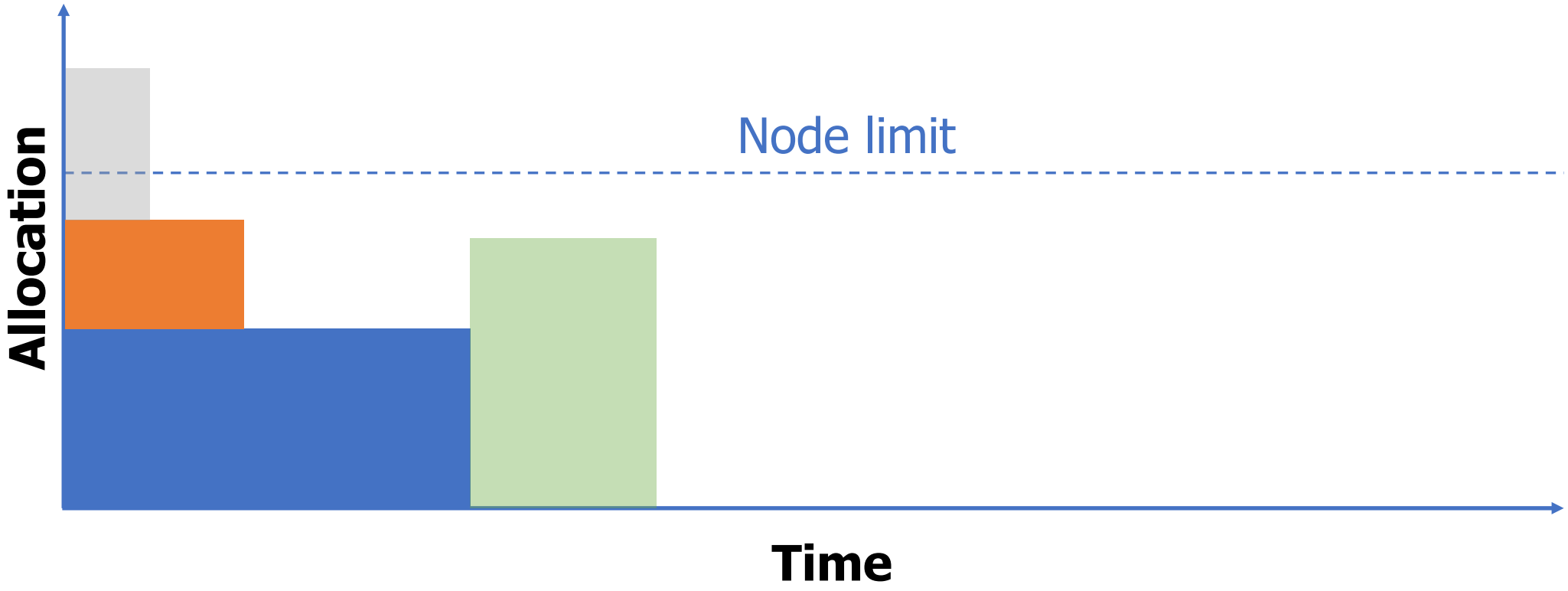
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



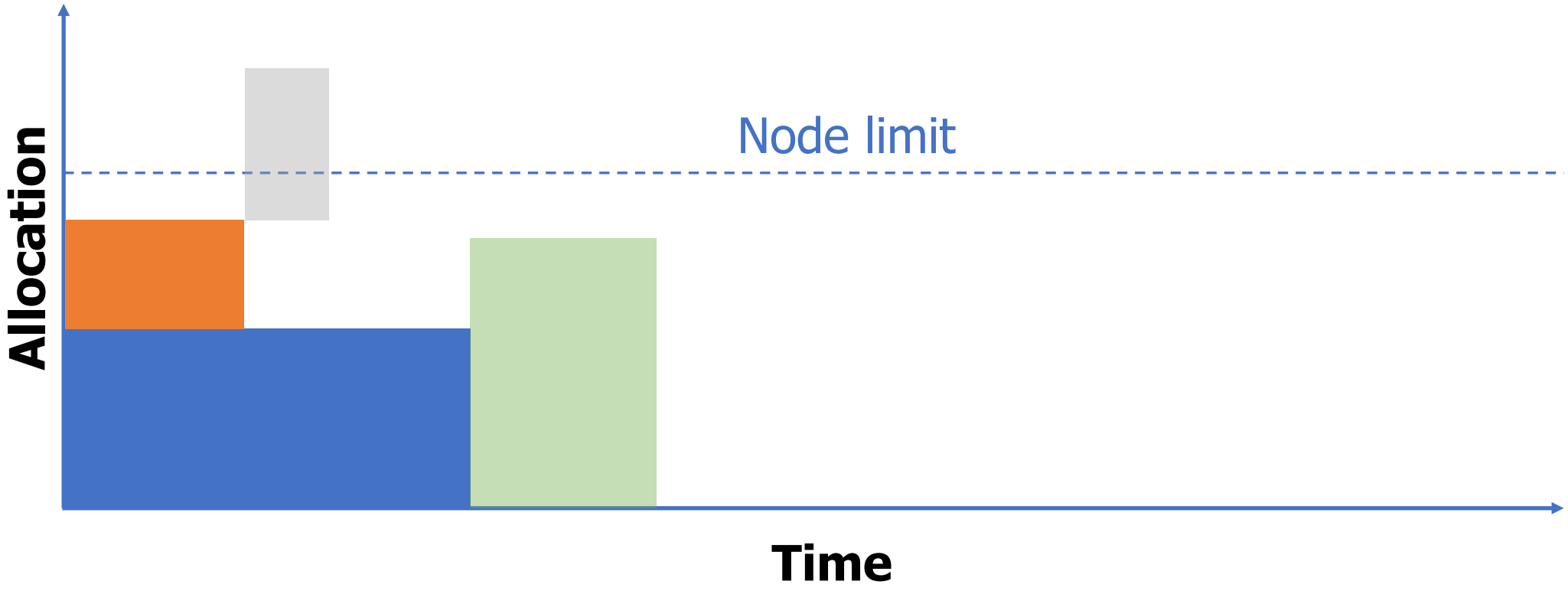
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



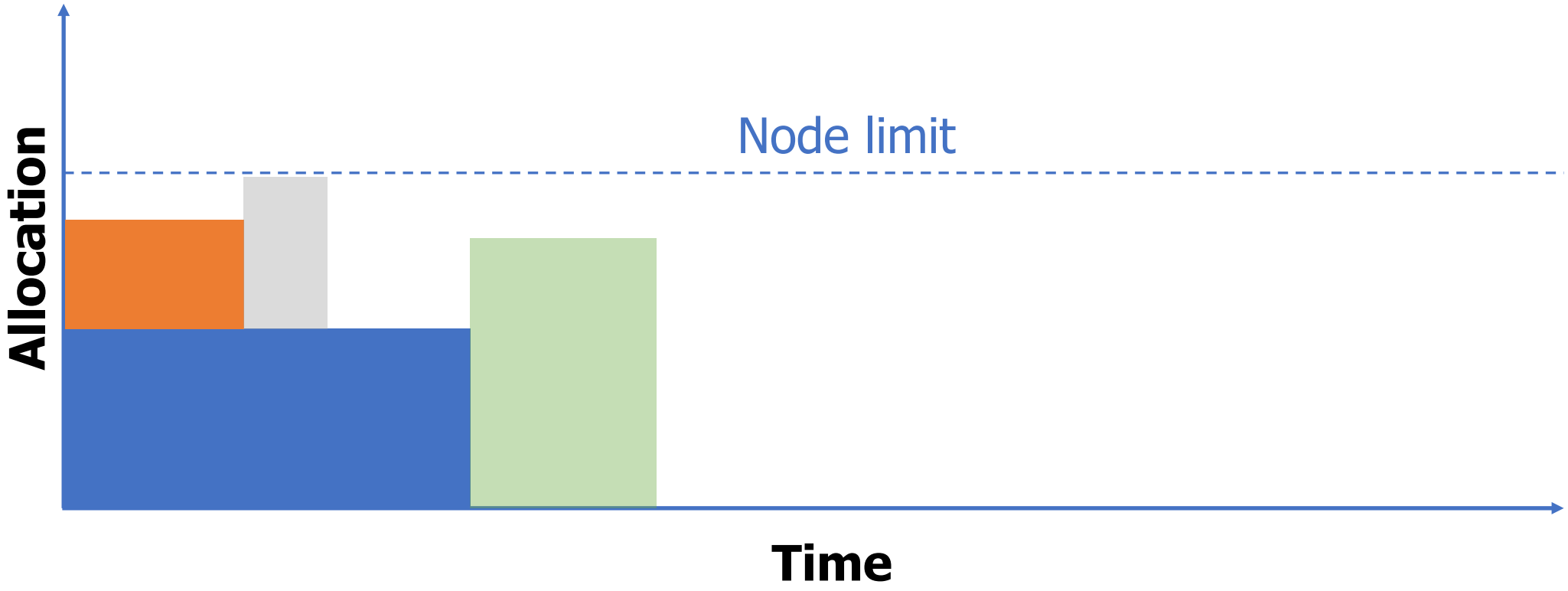
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



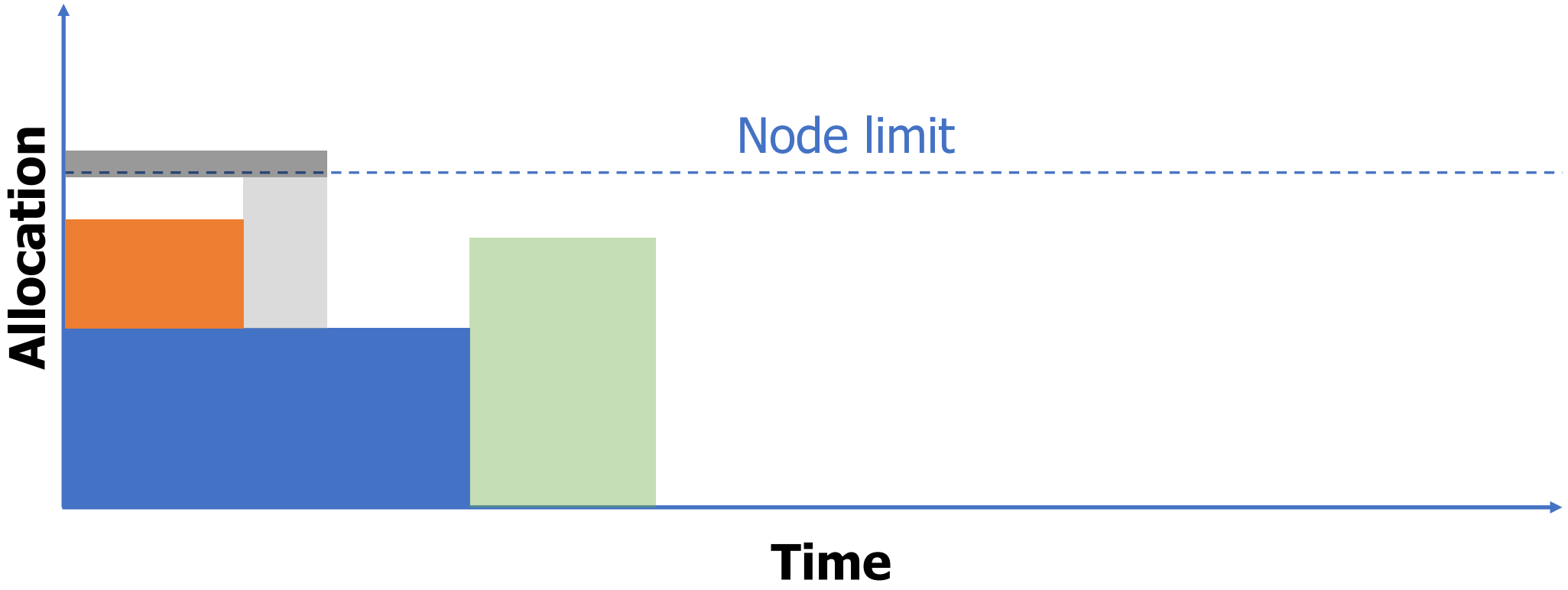
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



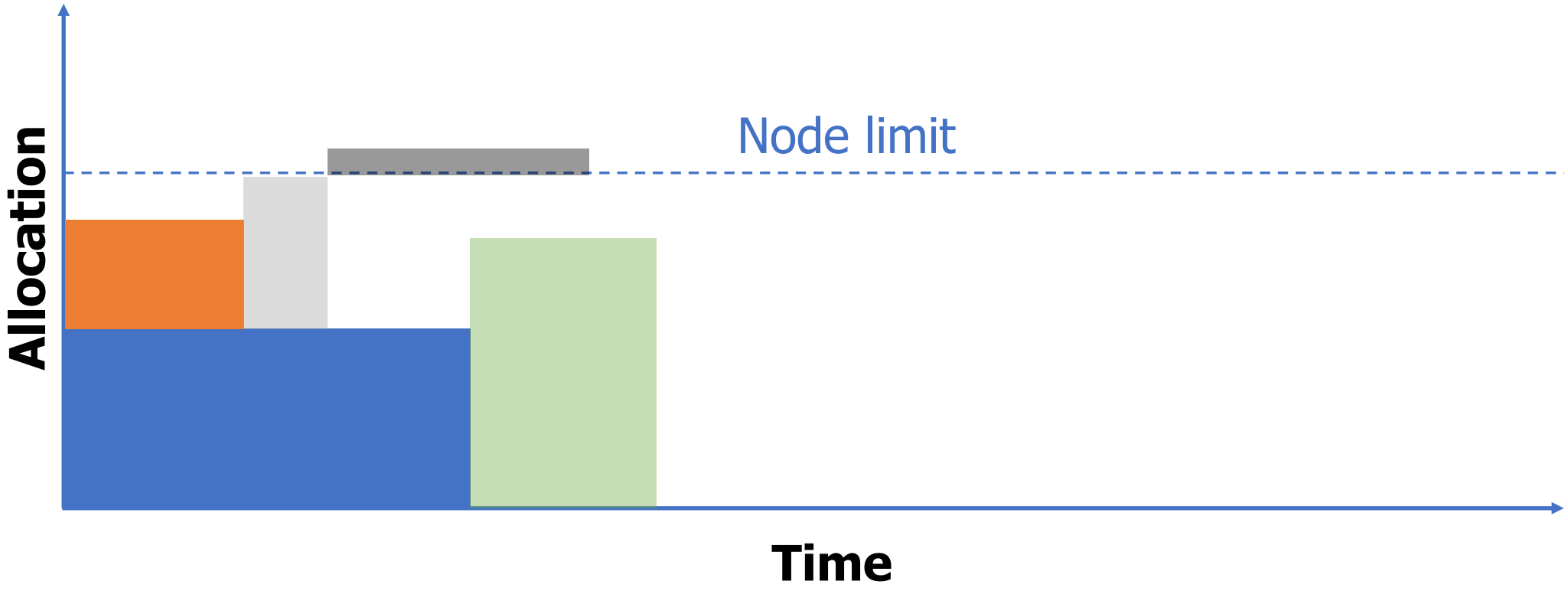
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



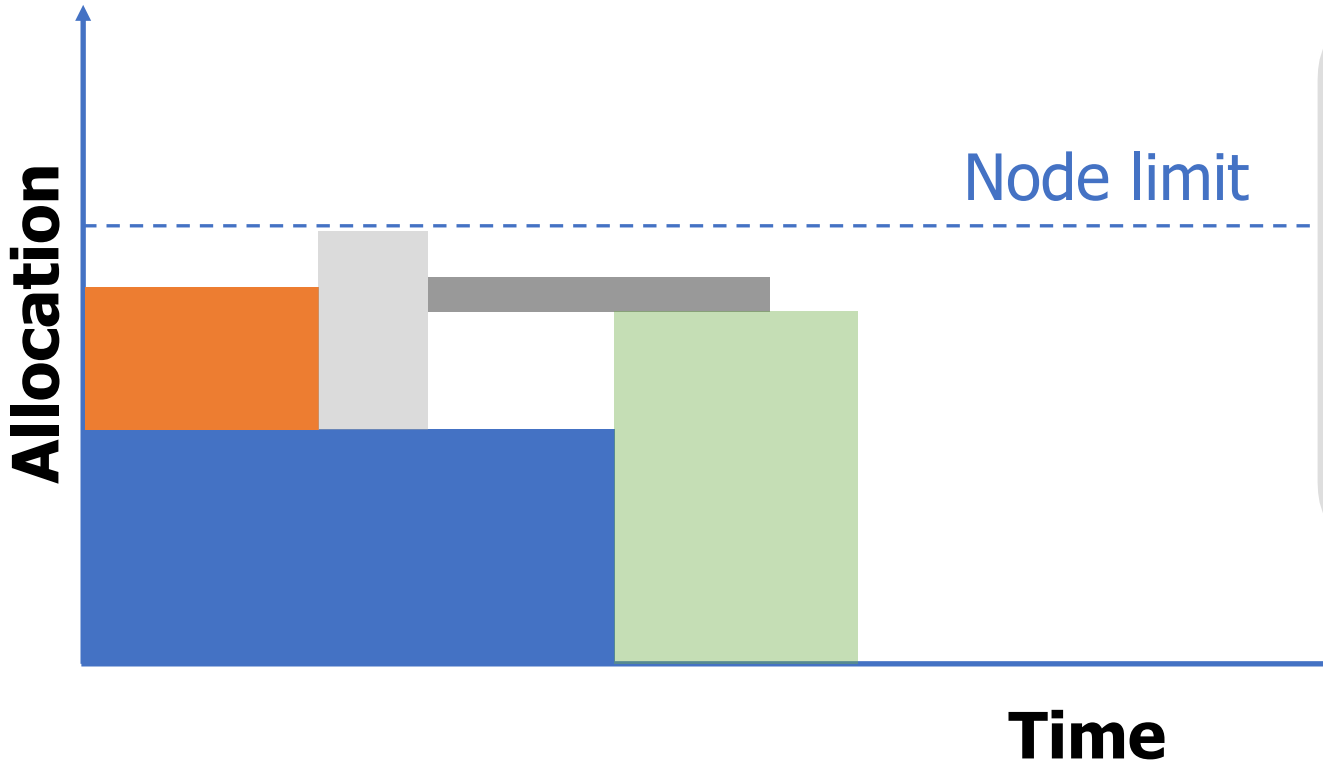
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



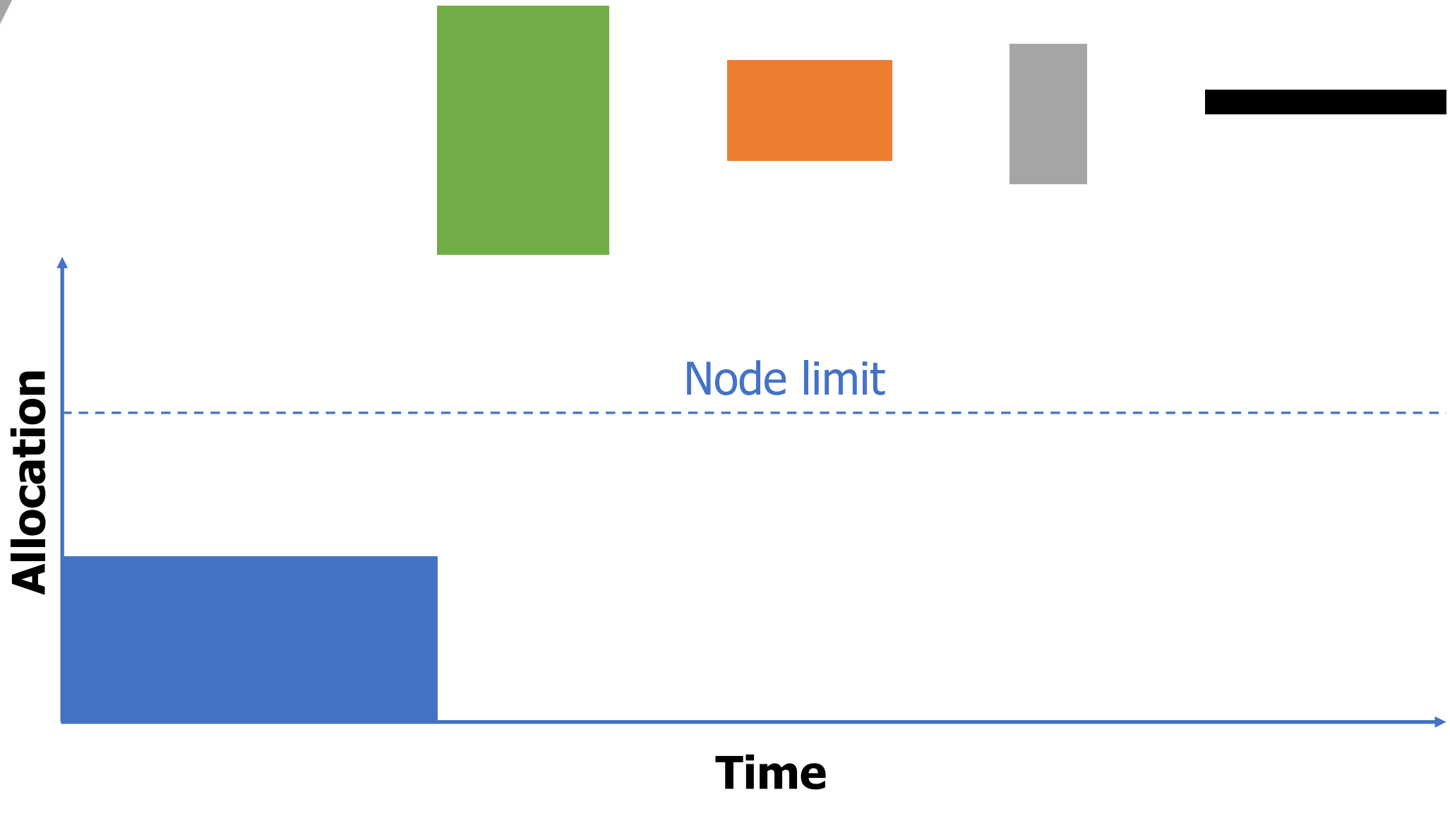
Just backfill (*JustBF*)

- If a job cannot run,
  reserve resources
  to prevent any further delays



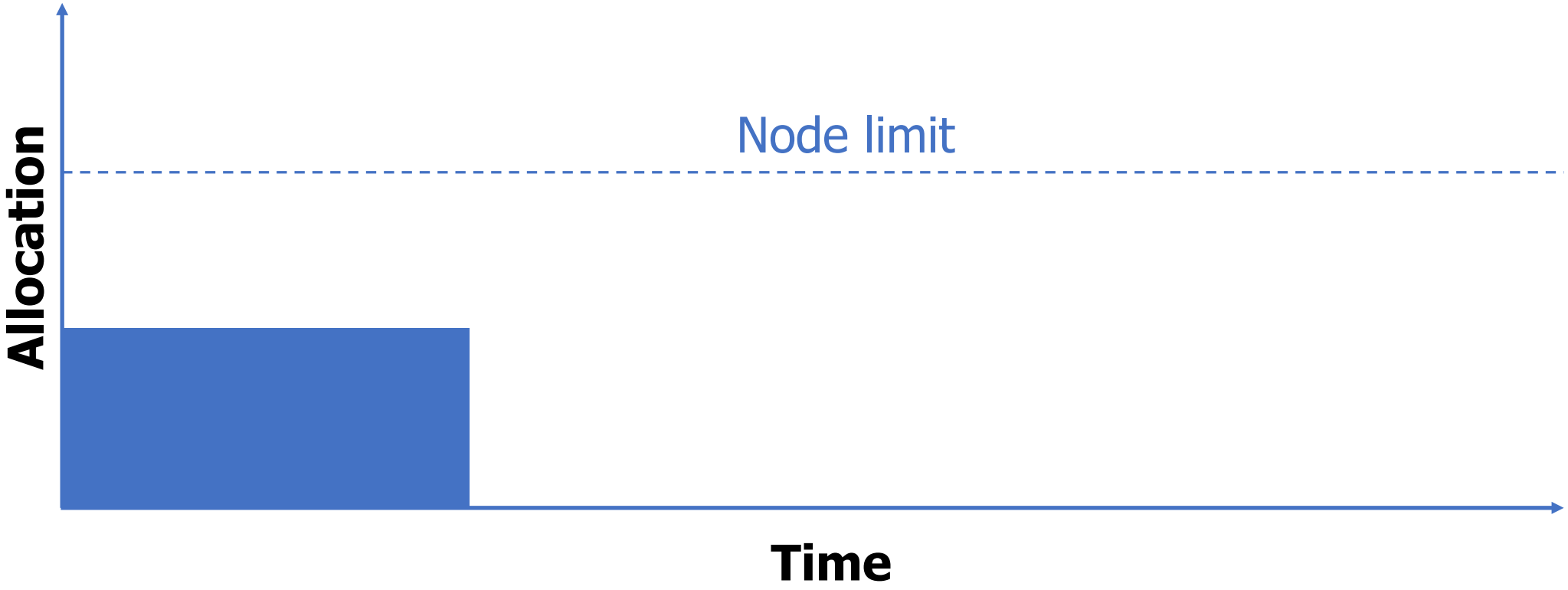
A later job may start
before an earlier job
but cannot delay it

Shortest Job First (*SJF*)-*JustBF*



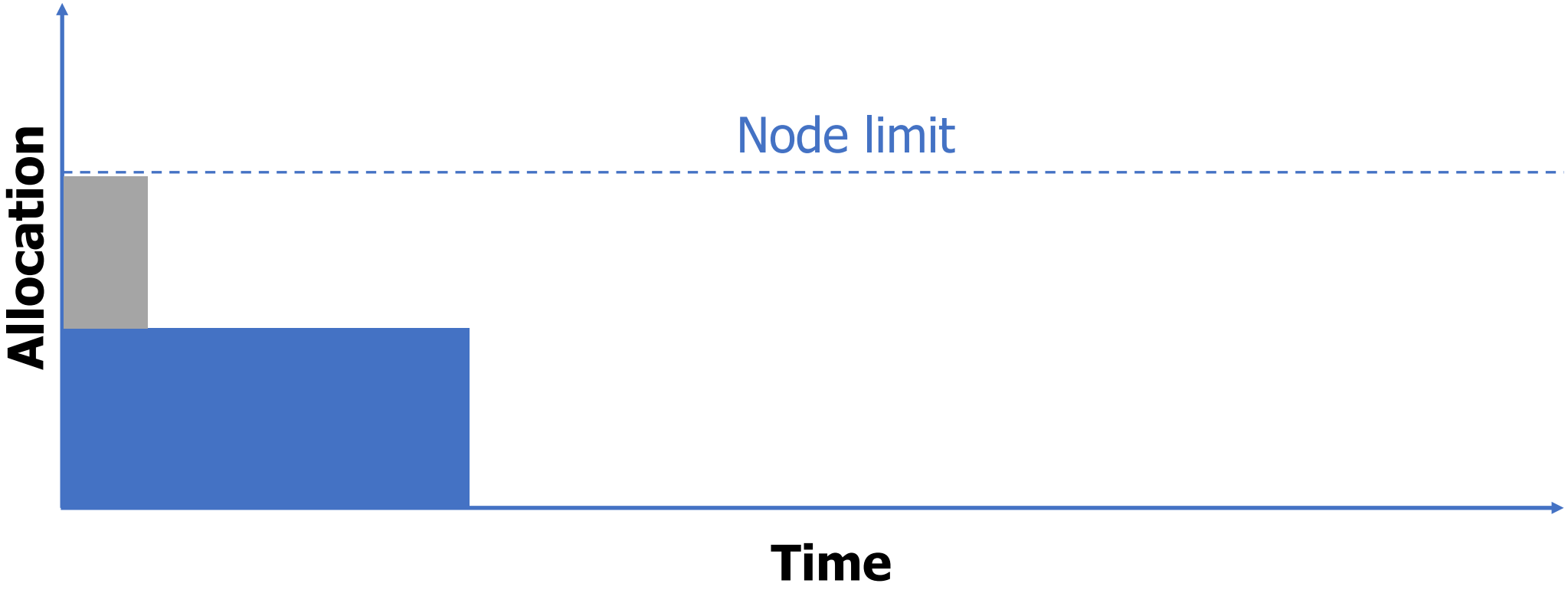
Shortest Job First (*SJF*)-*JustBF*

- Sort the queue



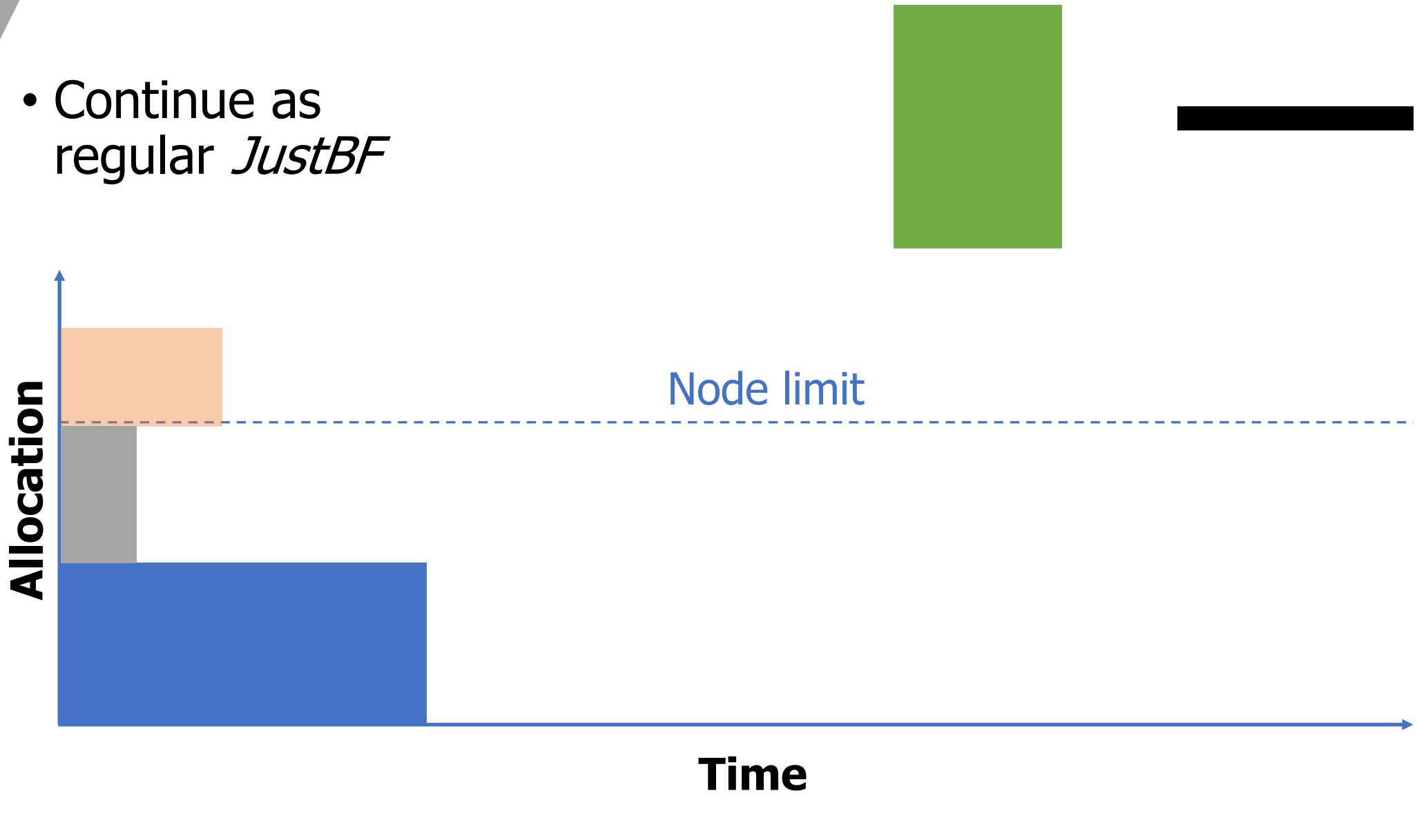
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



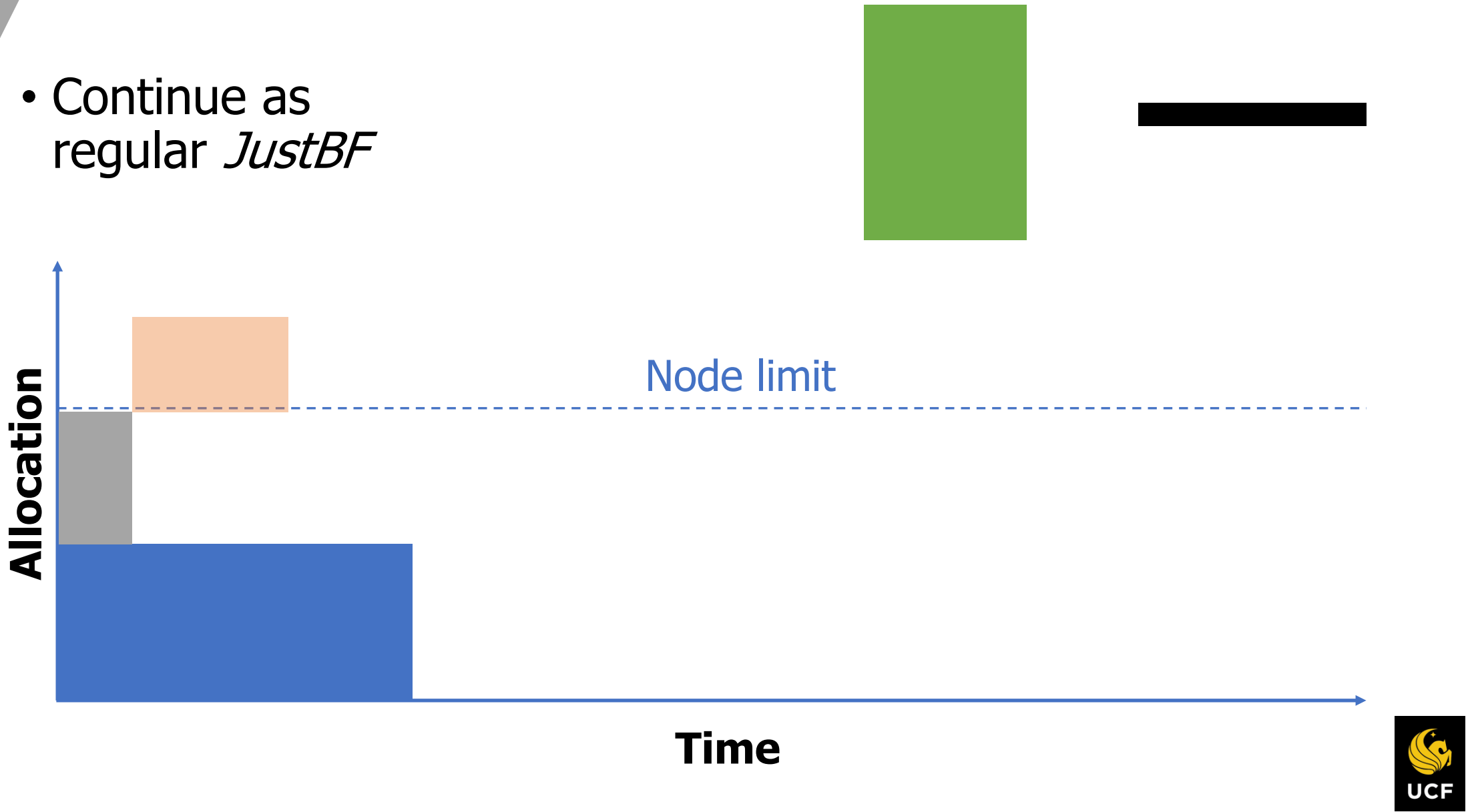
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



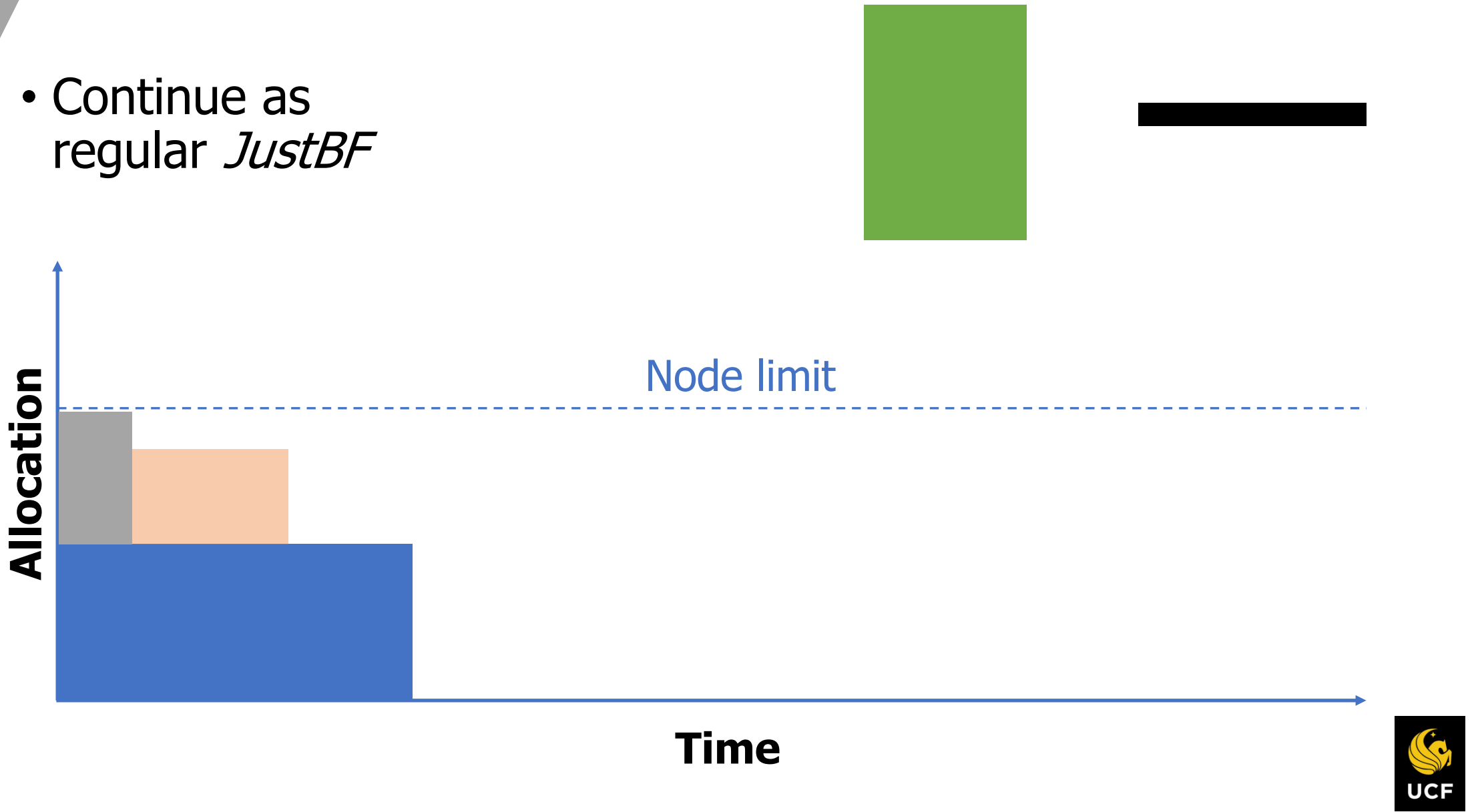
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



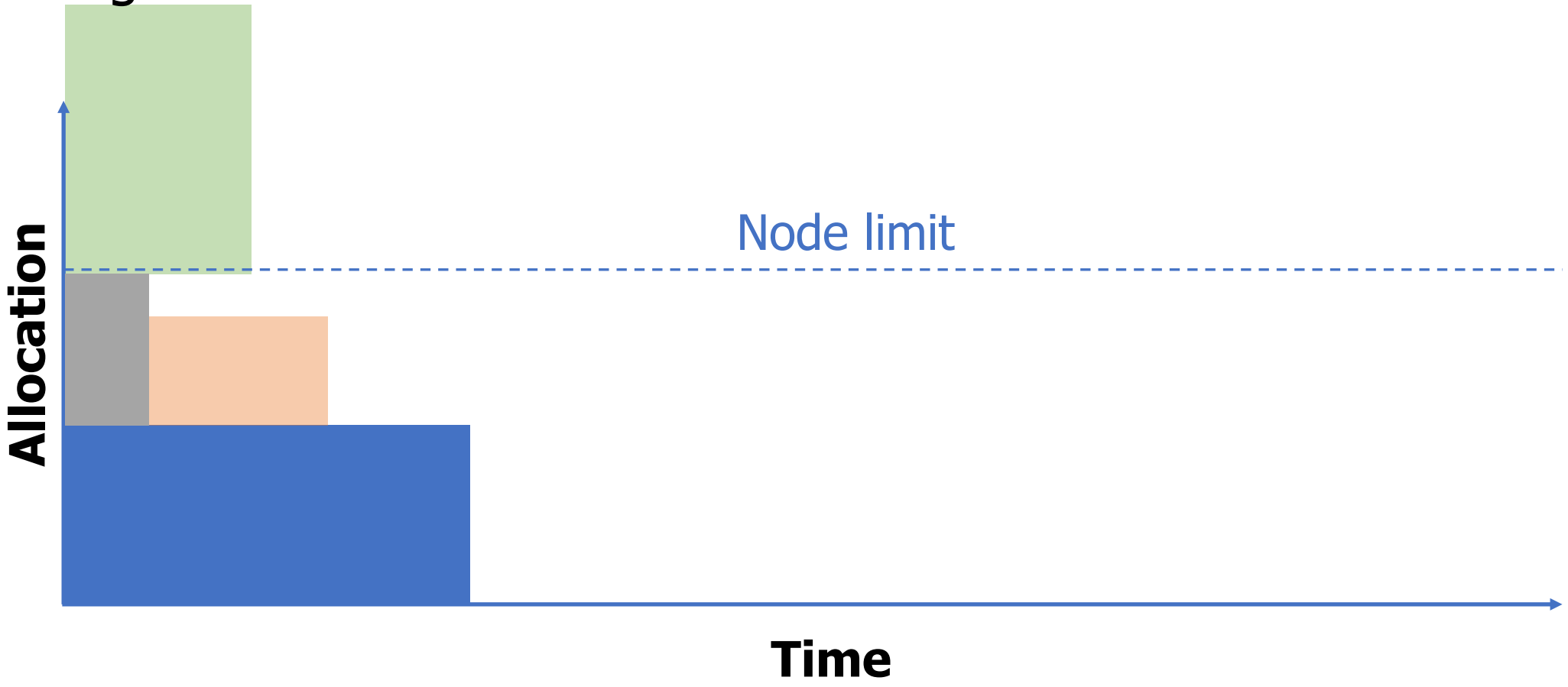
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



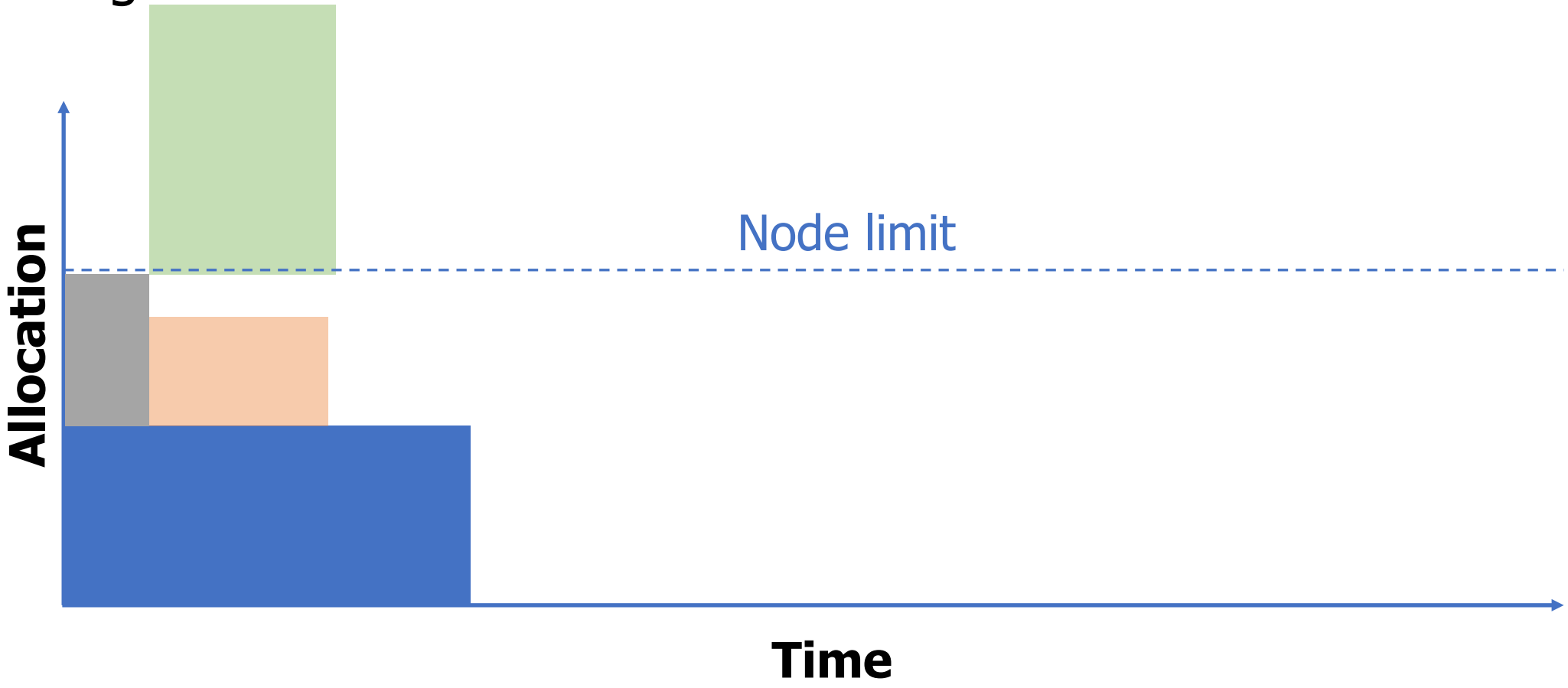
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



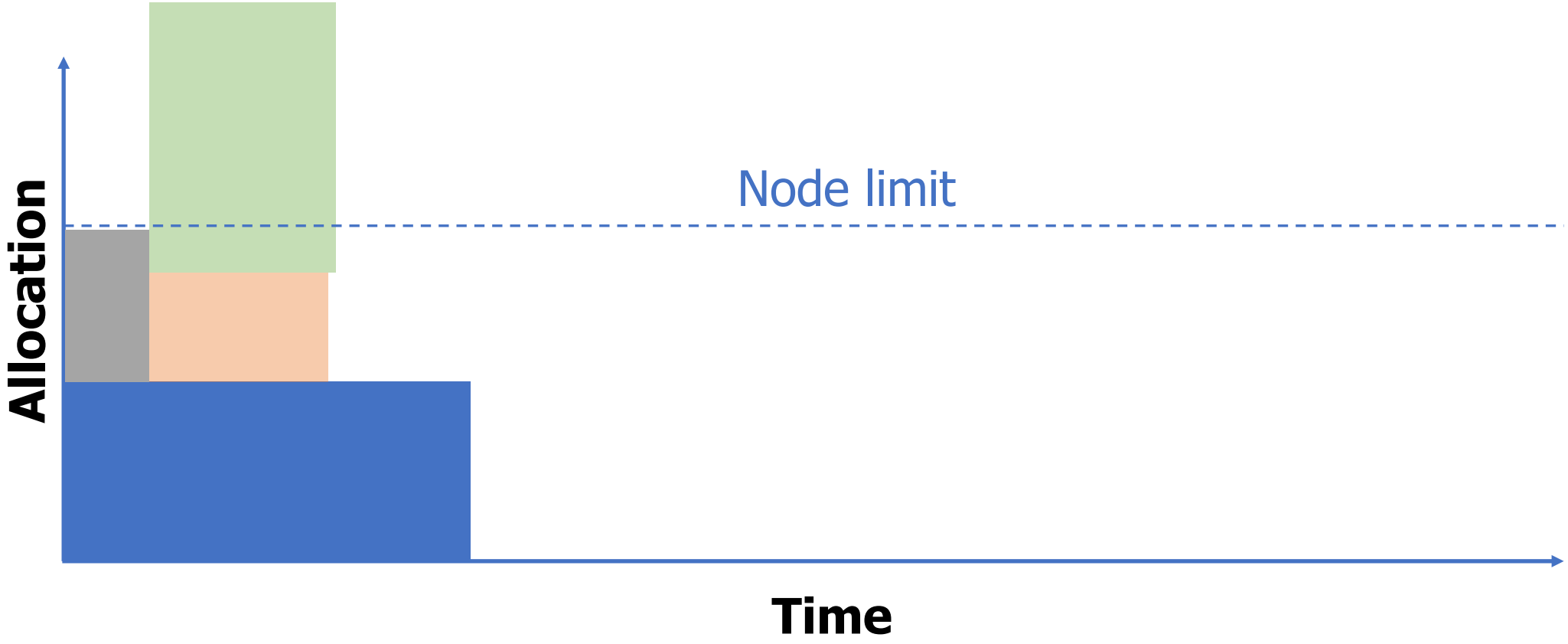
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



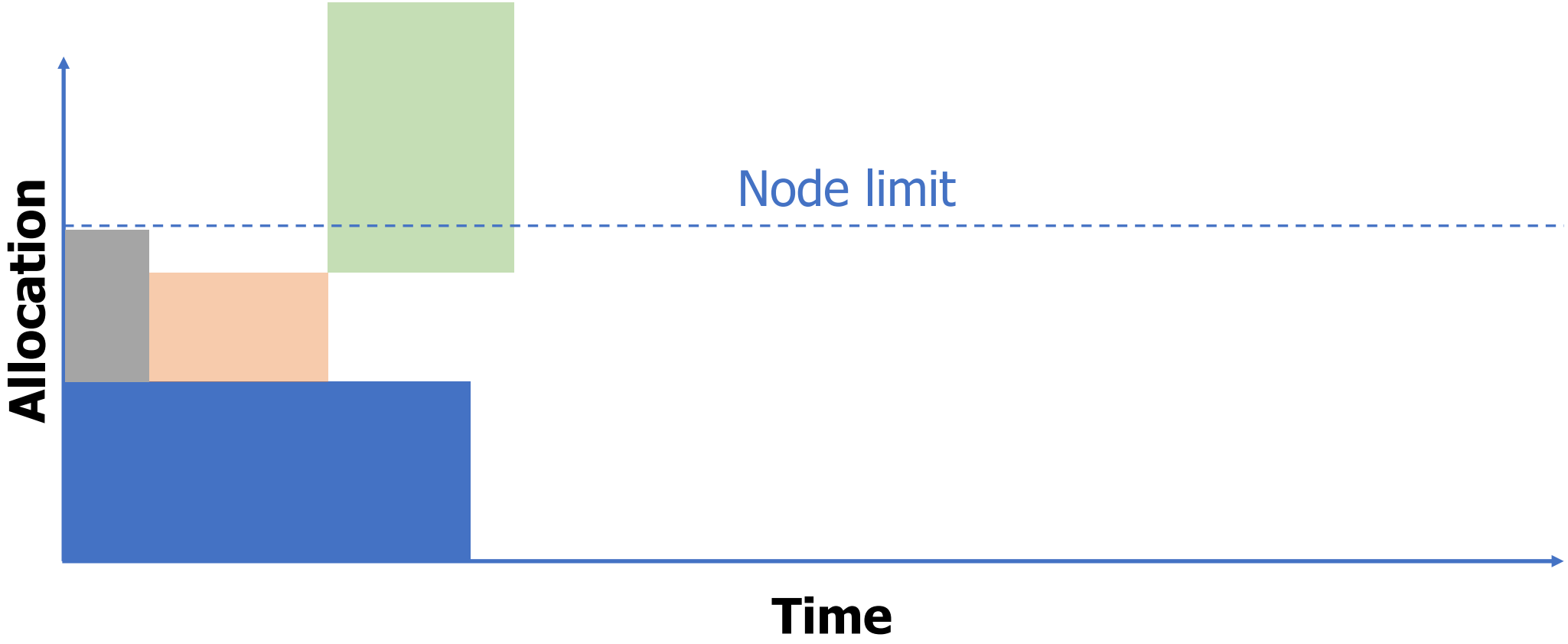
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



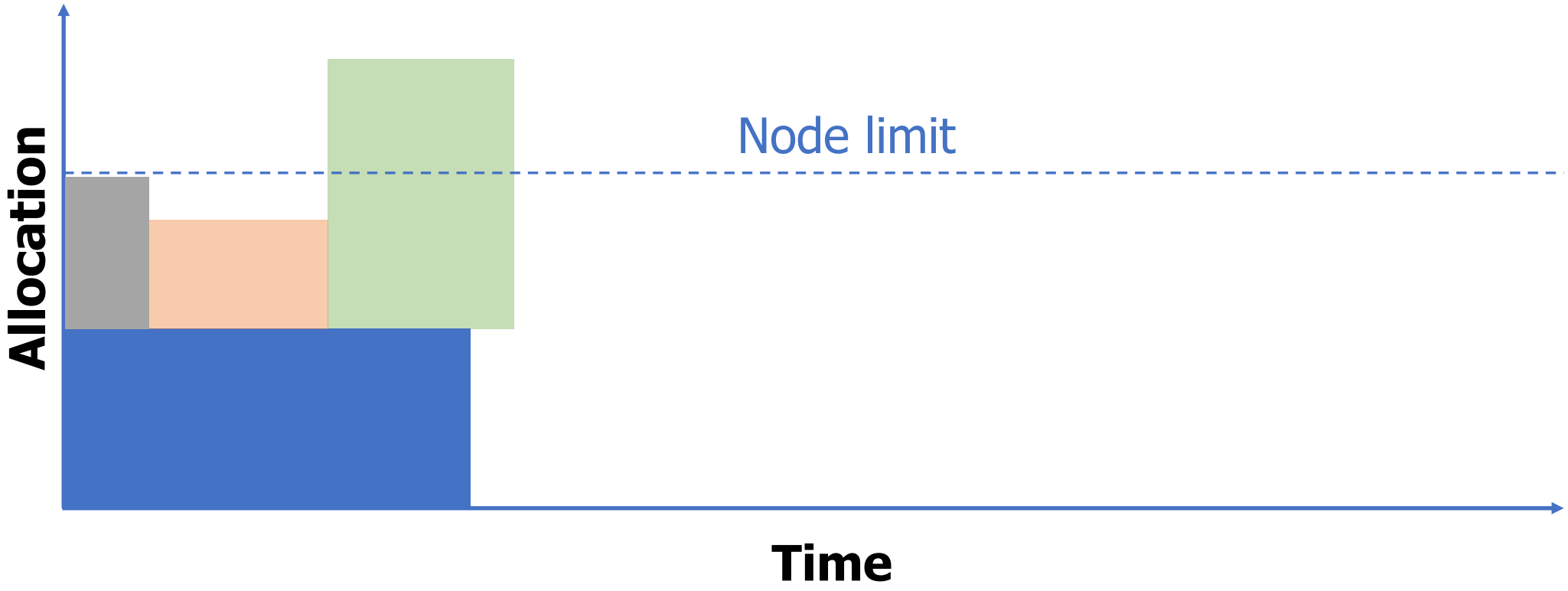
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



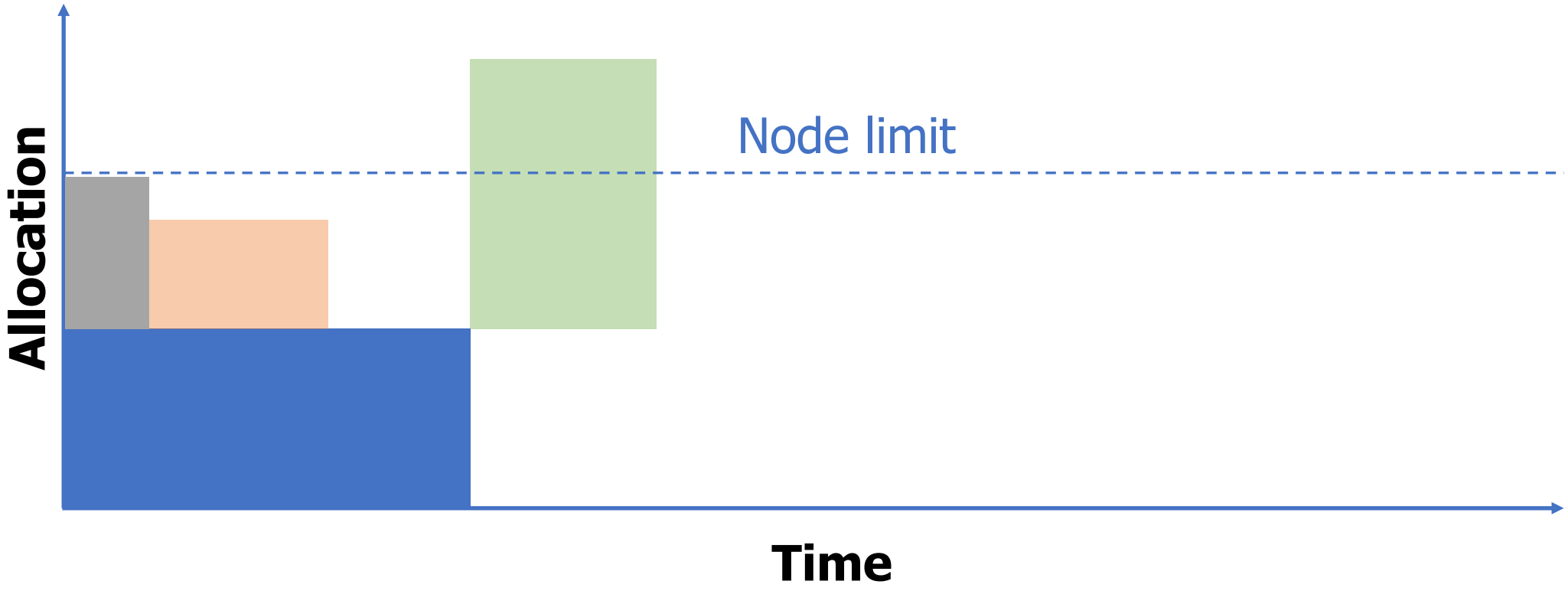
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



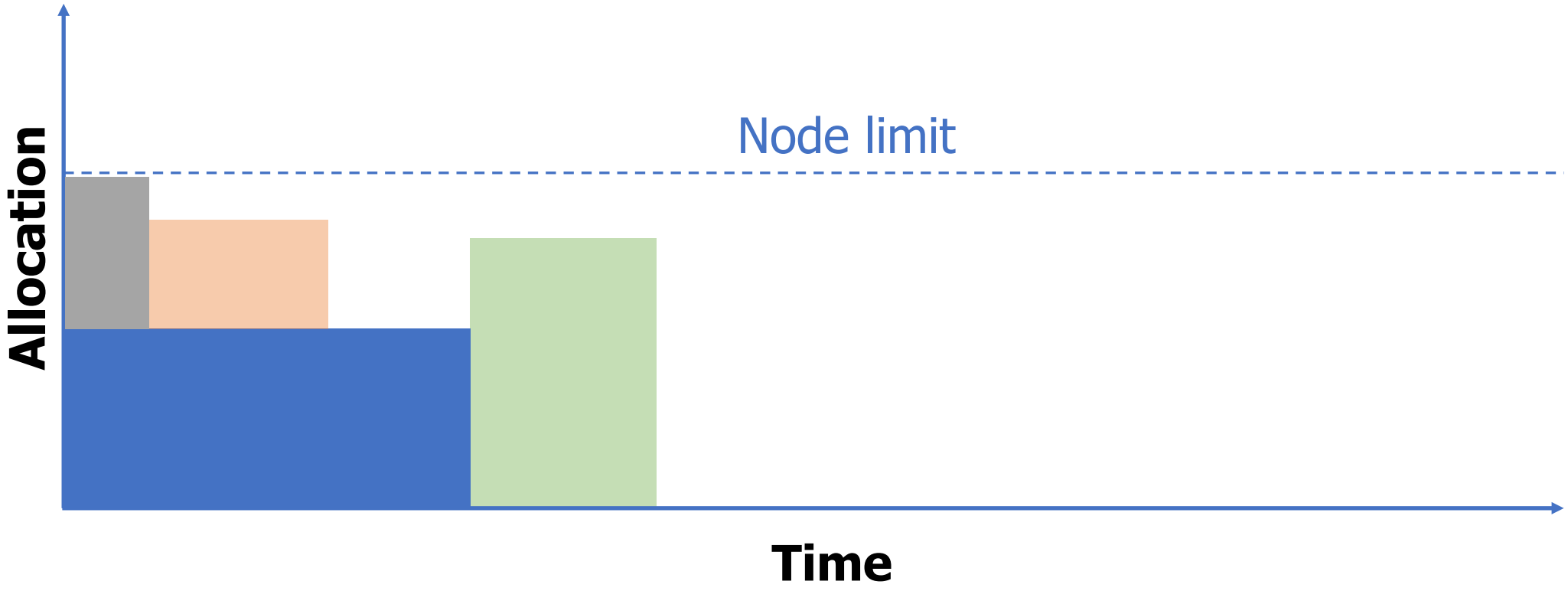
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



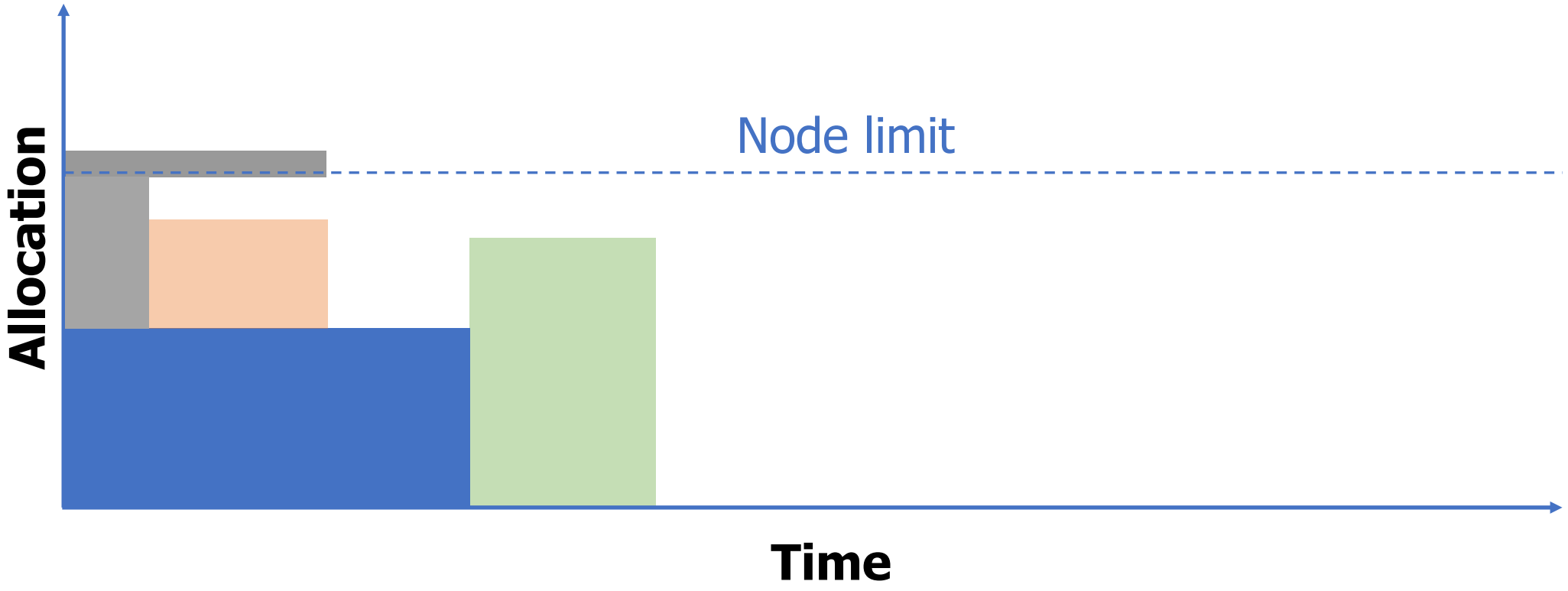
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



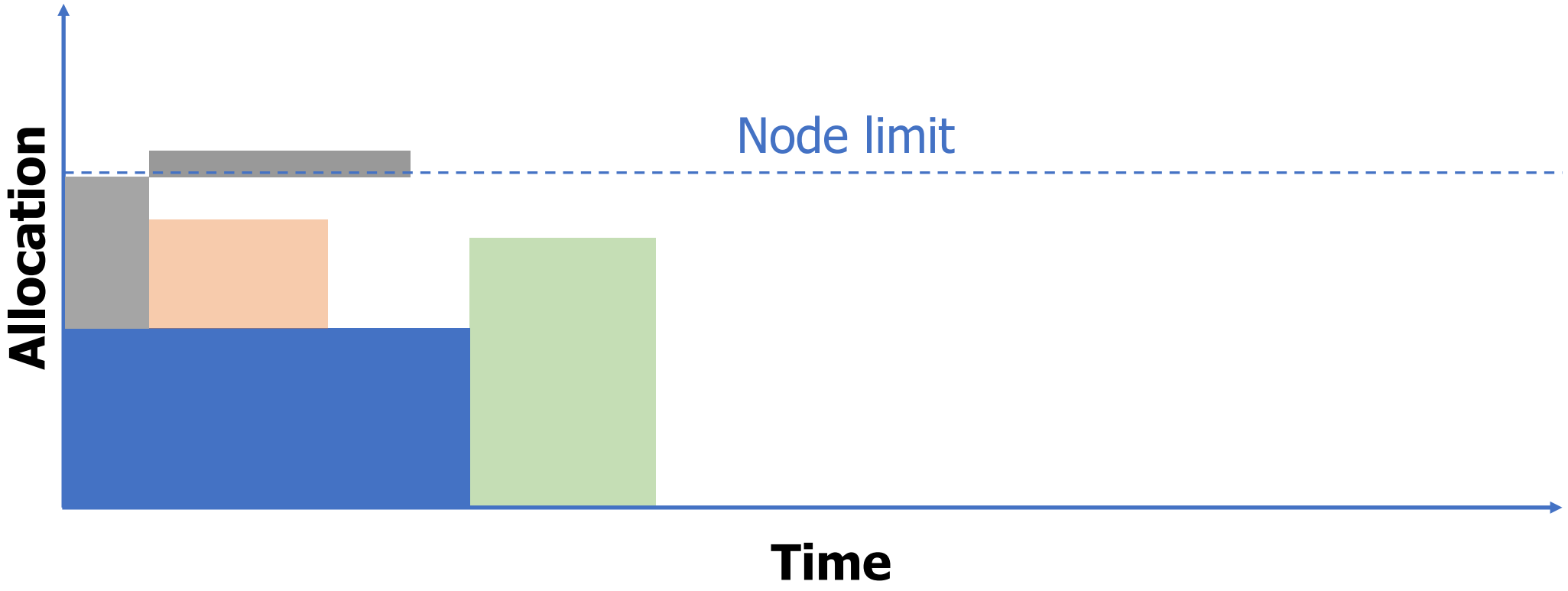
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



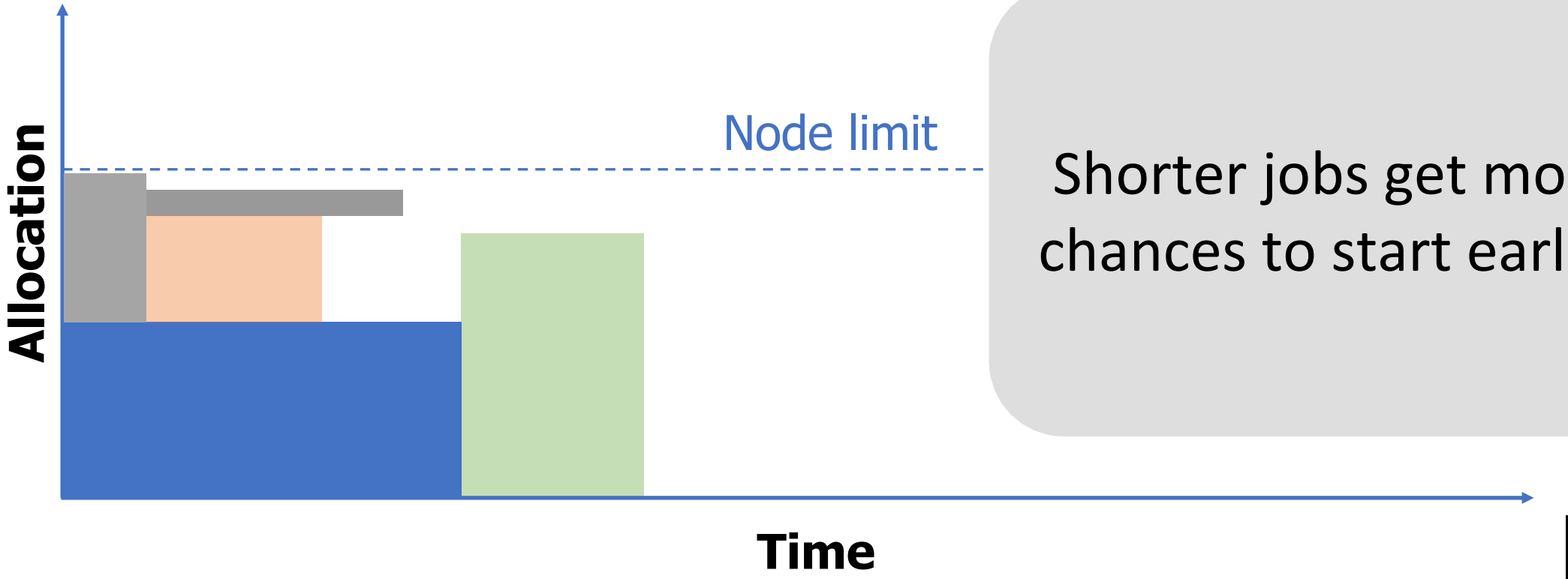
Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



Shortest Job First (*SJF*)-*JustBF*

- Continue as regular *JustBF*



Shorter jobs get more chances to start earlier

Sort orders

Default : No reorder (First-Come-First-Serve)

SJF : Shortest Job First

- ascending order according to estimated runtime
  - *sometimes we assume that we know real runtime*

SAF : Smallest Area First

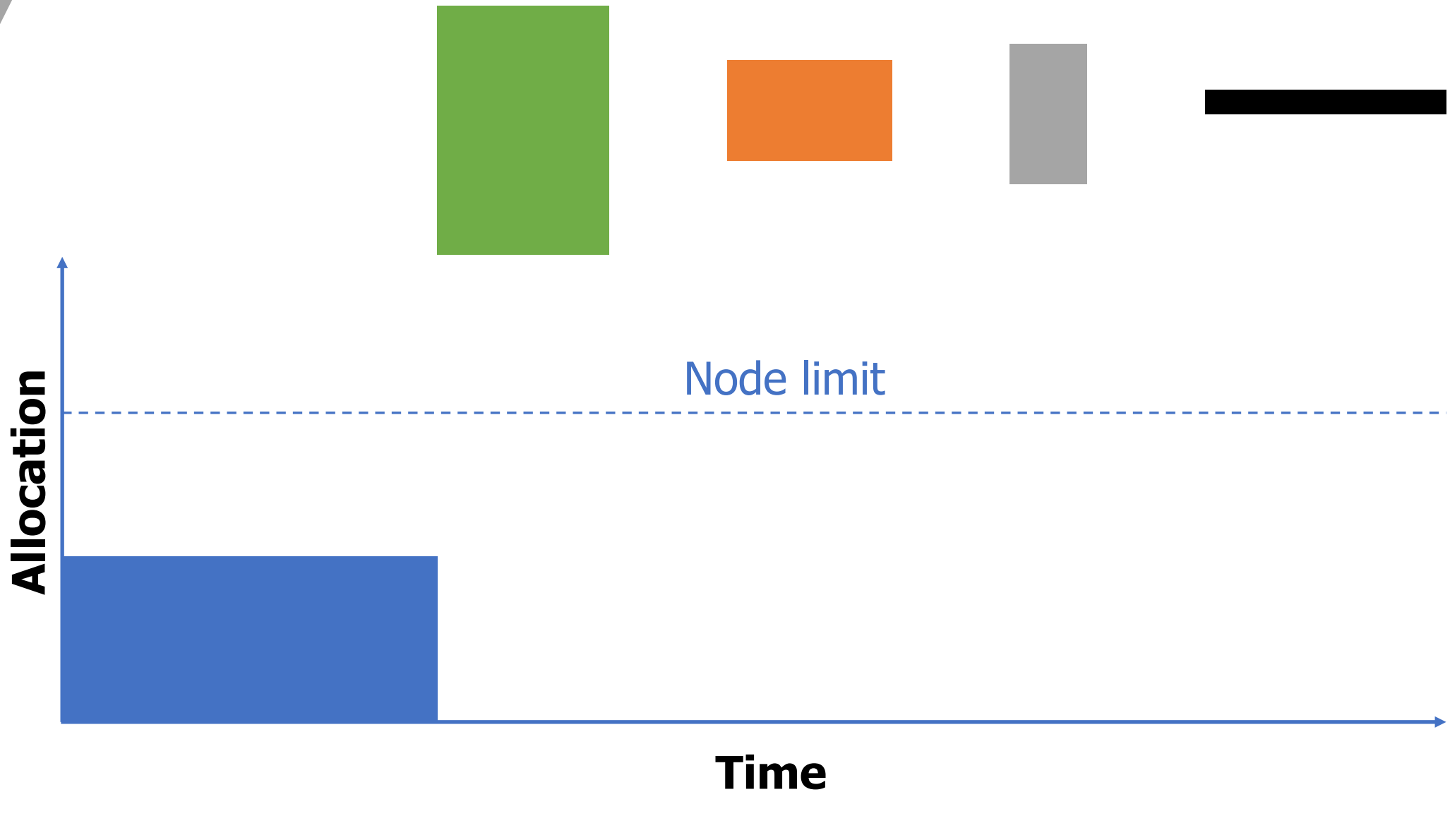
- ascending order according to job area
- job area is $r_j E_j$
  - r_j is resource requirement
  - E_j is estimated runtime LAF: Largest Area First

LAF : Largest Area First

- descending order according to job area

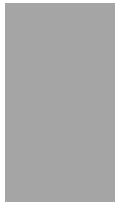
UCF

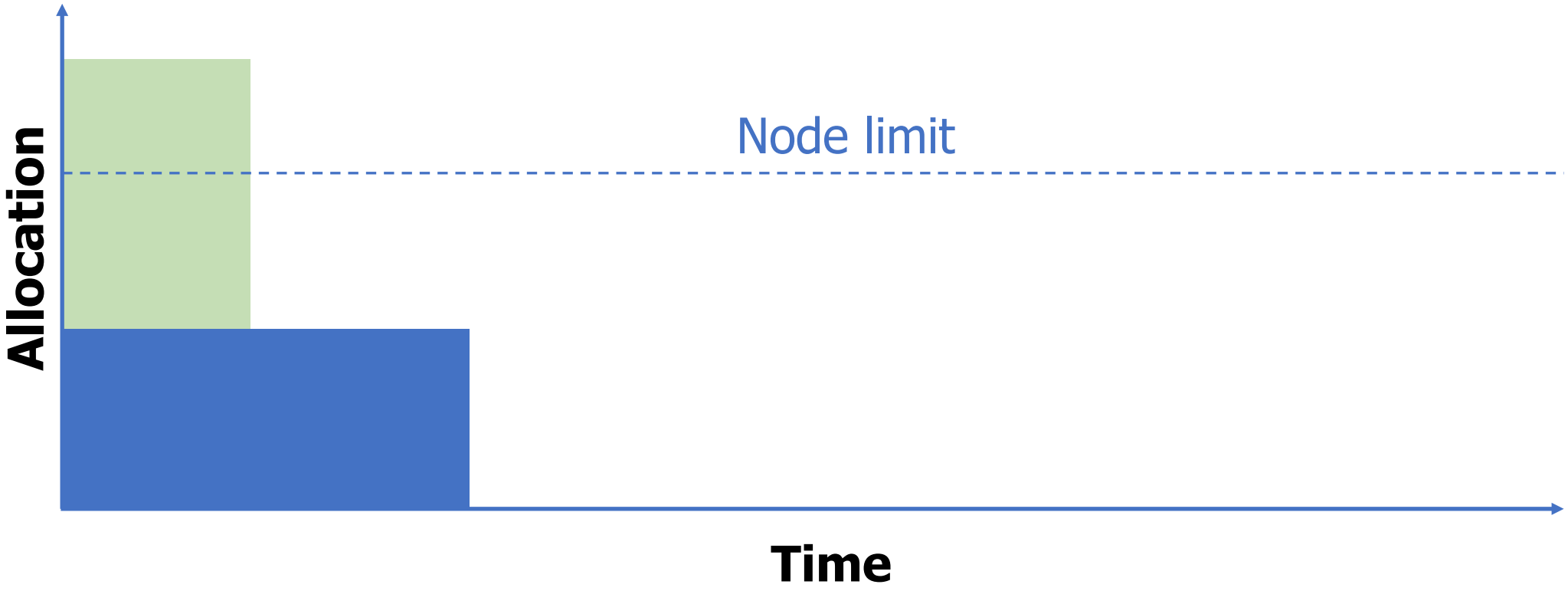
EASY backfill



EASY backfill

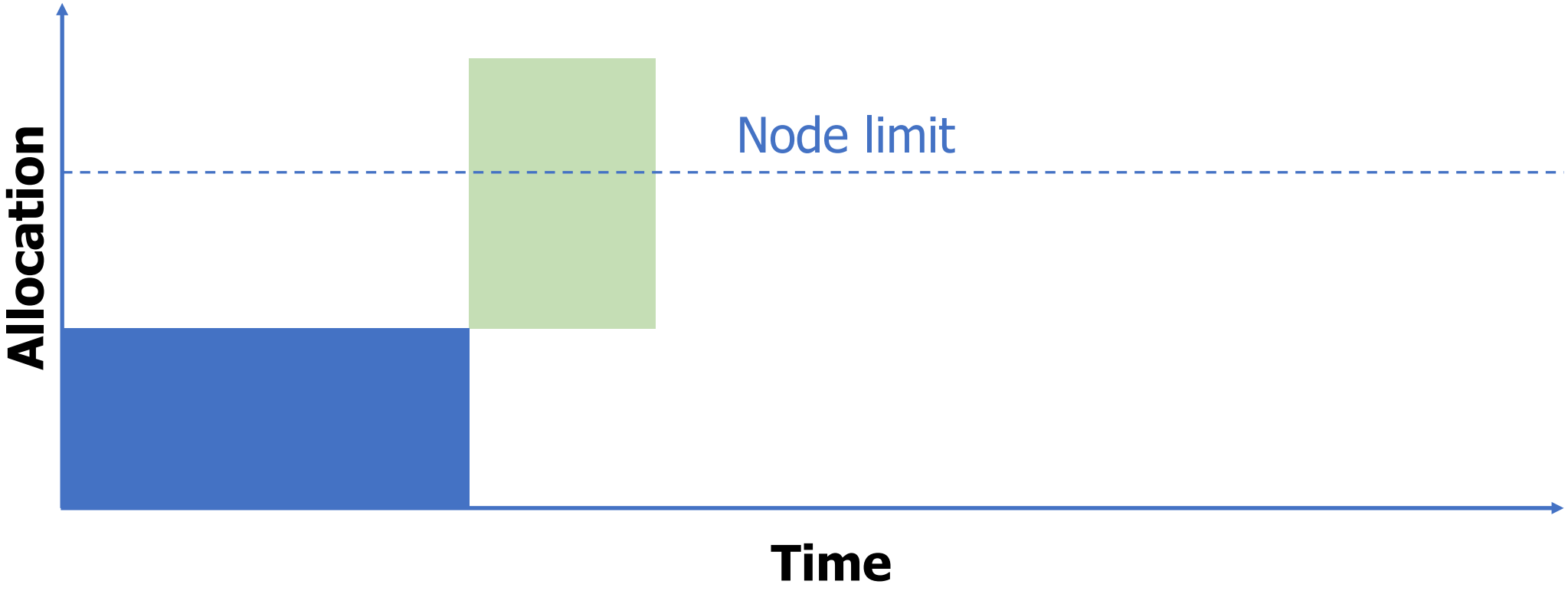
- Make only one reservation per scheduling round

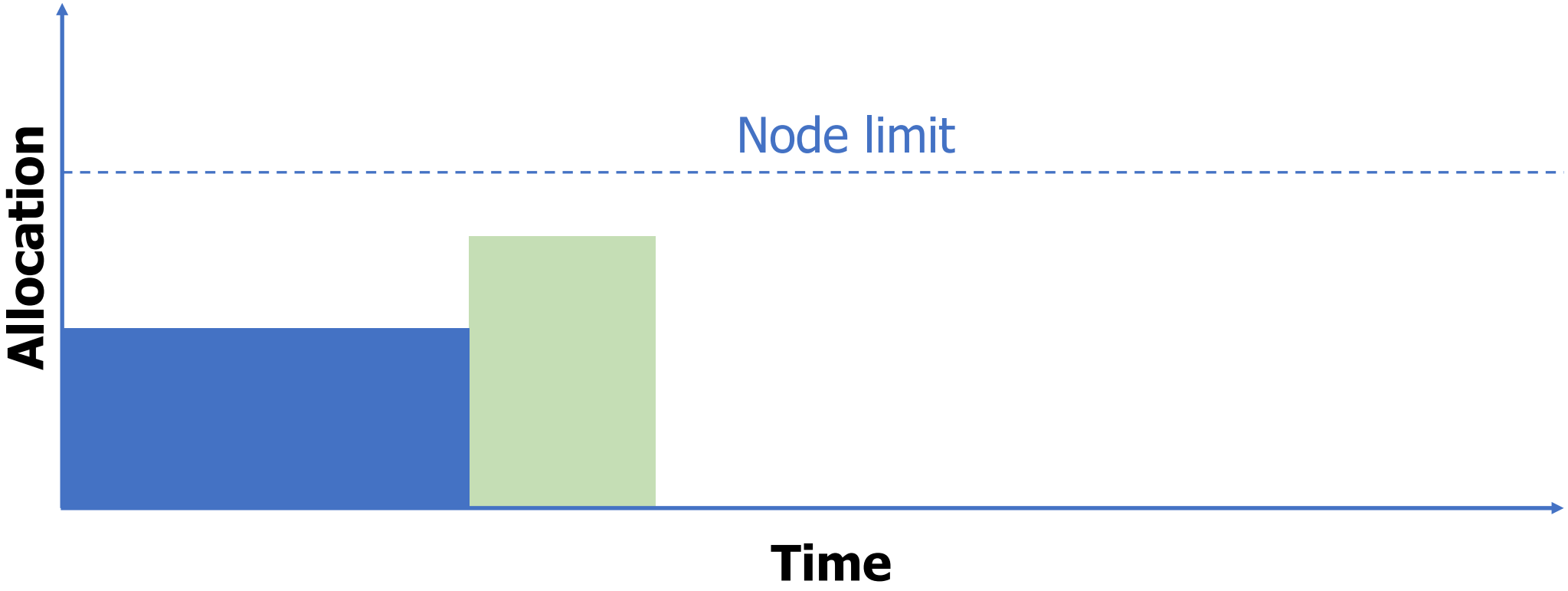
EASY backfill

- Make only one reservation per scheduling round



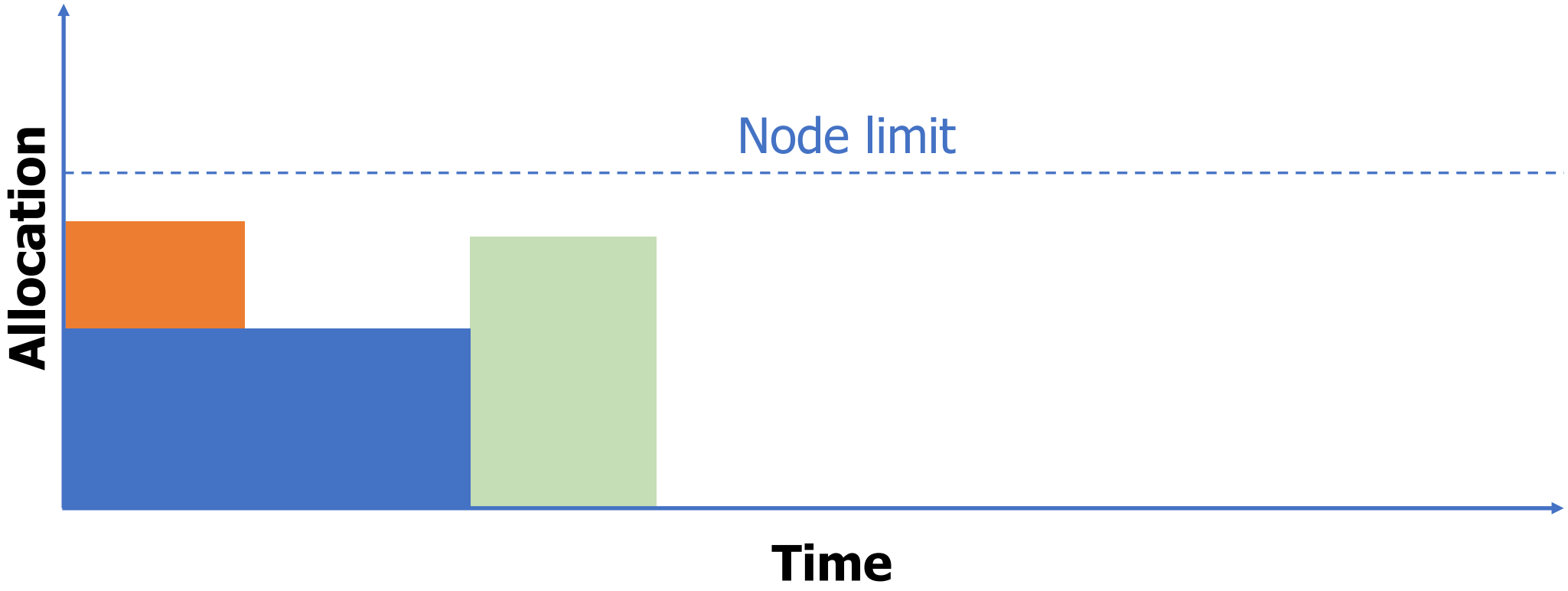
EASY backfill

- Make only one reservation per scheduling round



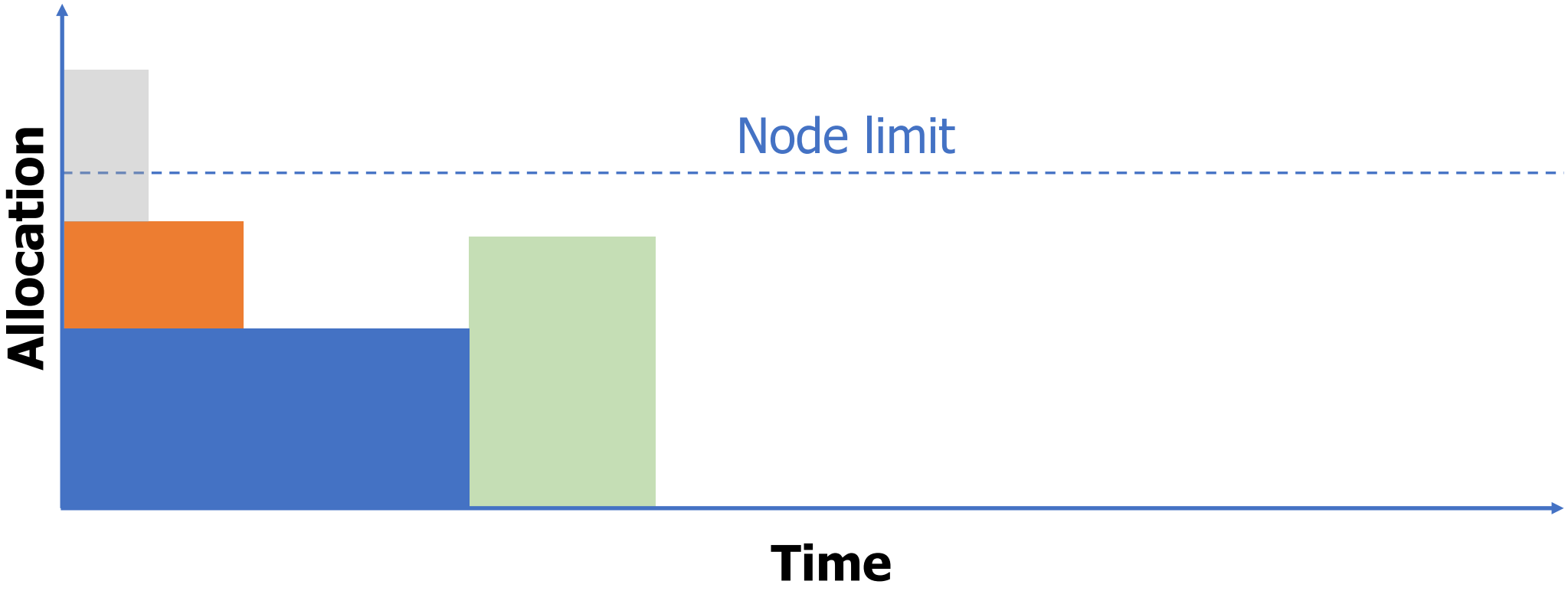
EASY backfill

- Continue "aggressive" scheduling



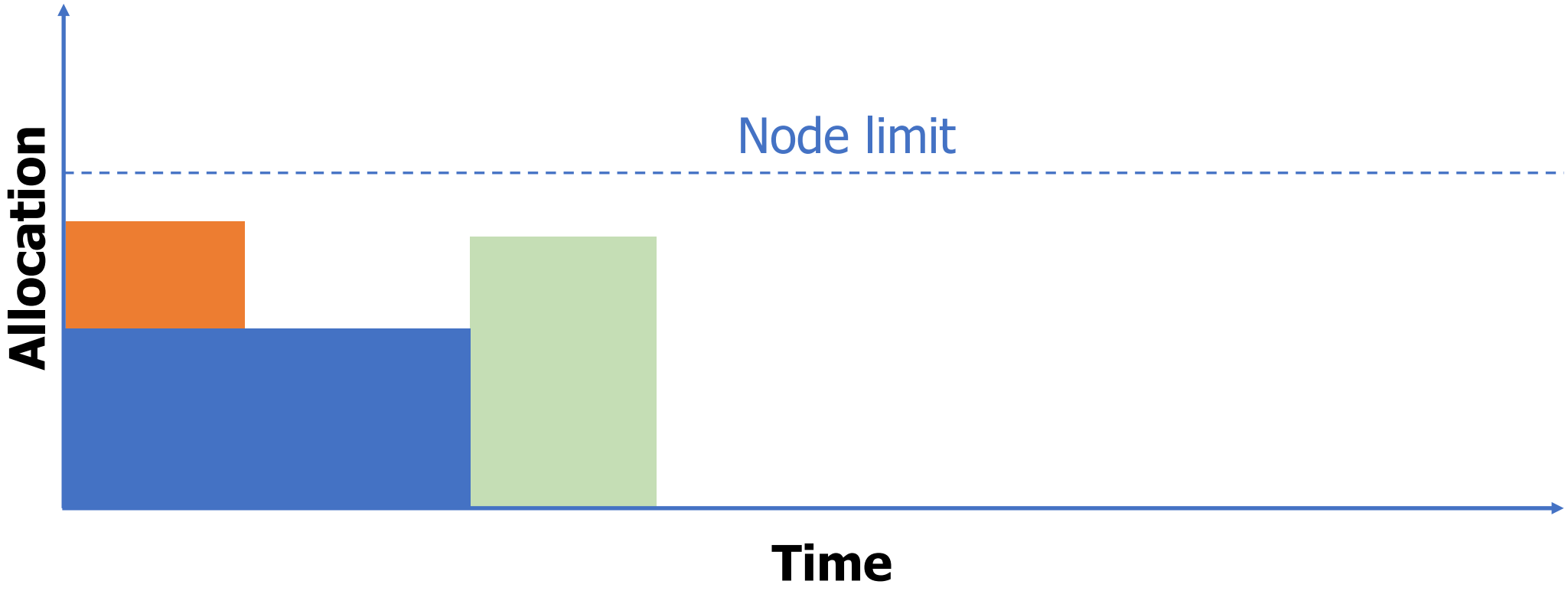
EASY backfill

- Continue "aggressive" scheduling



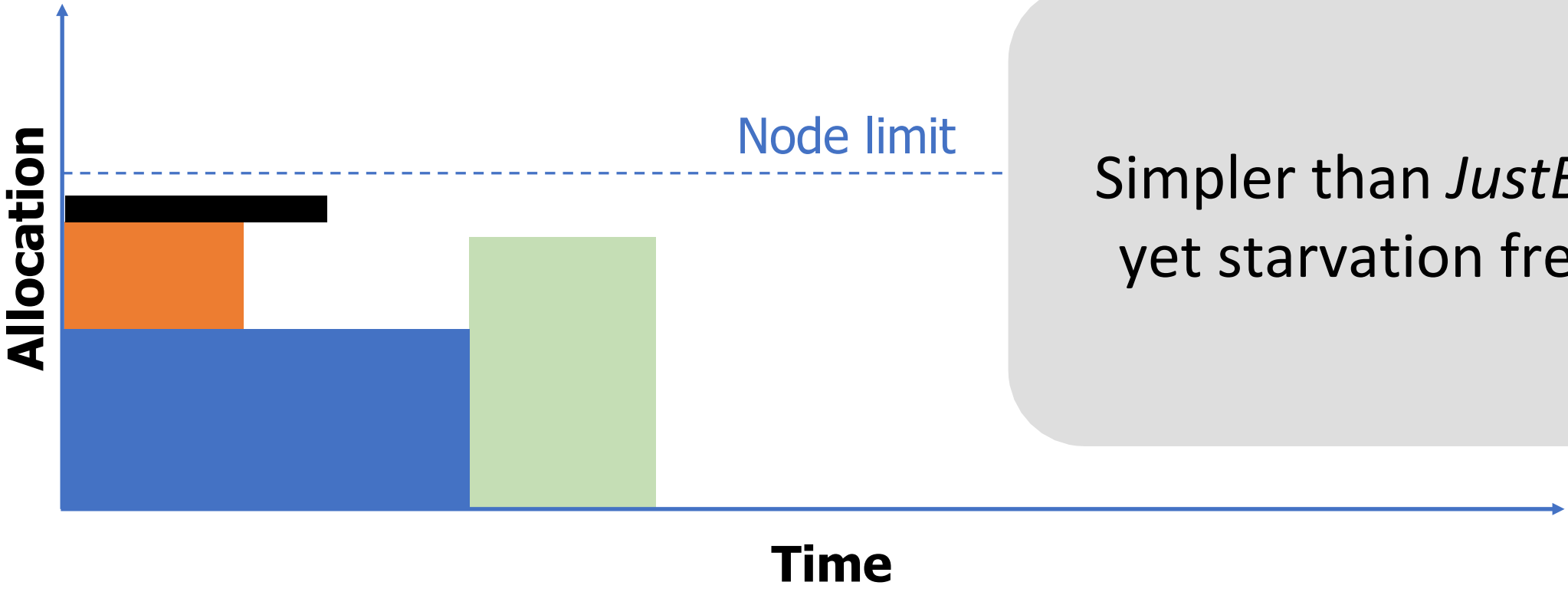
EASY backfill

- Continue "aggressive" scheduling



EASY backfill

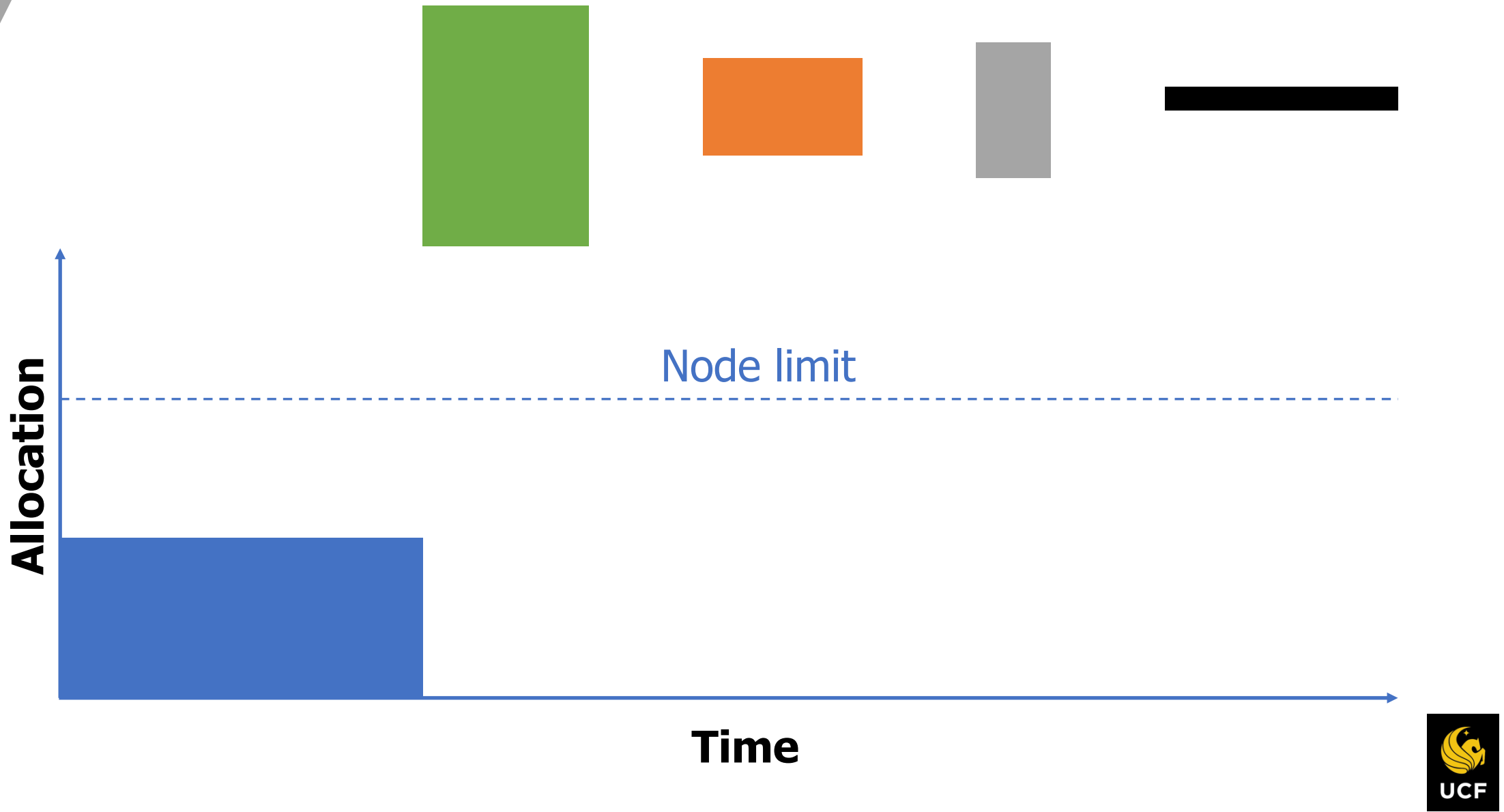
- Continue "aggressive" scheduling



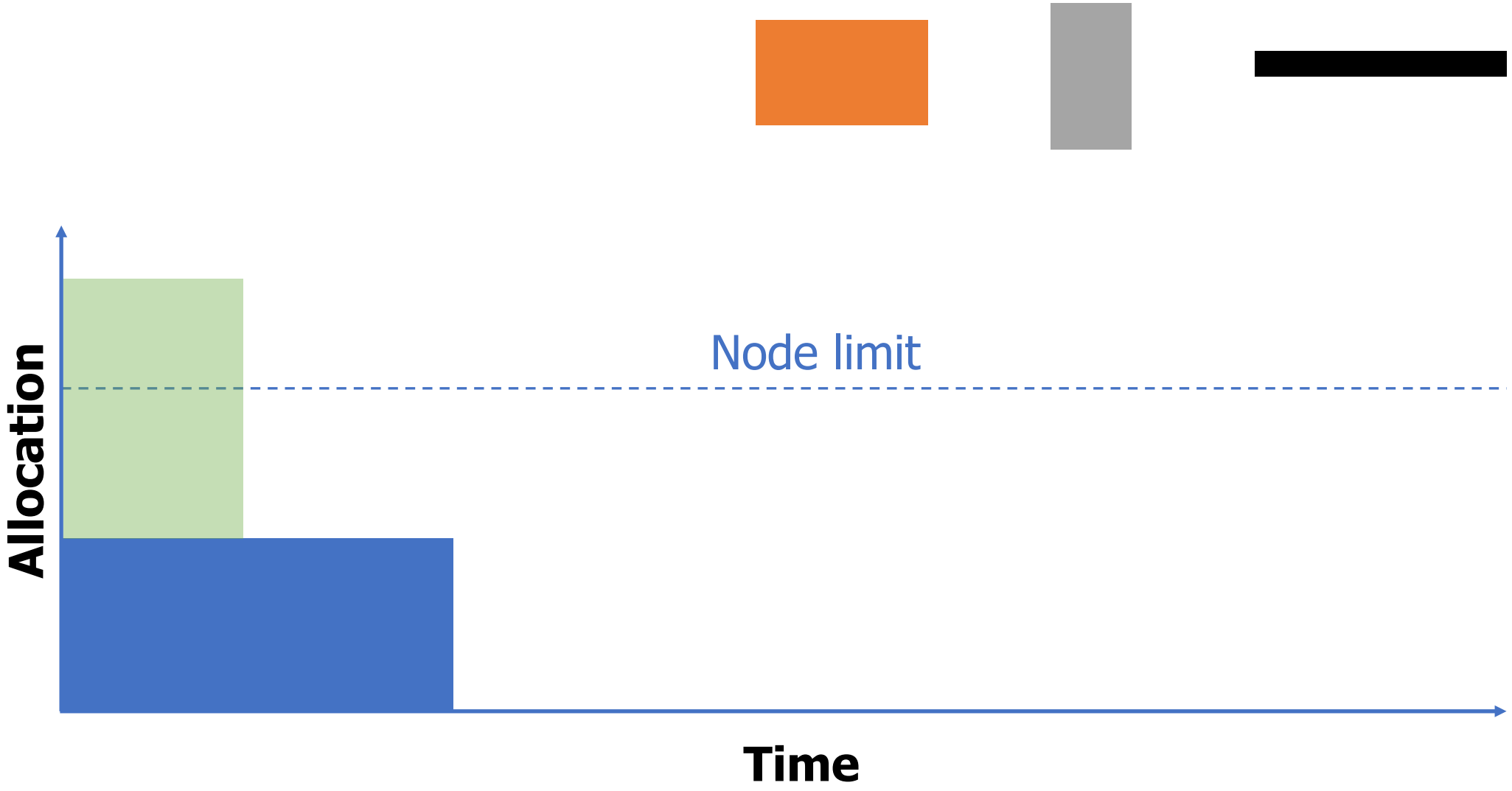
Simpler than *JustBF,* yet starvation free

UCF

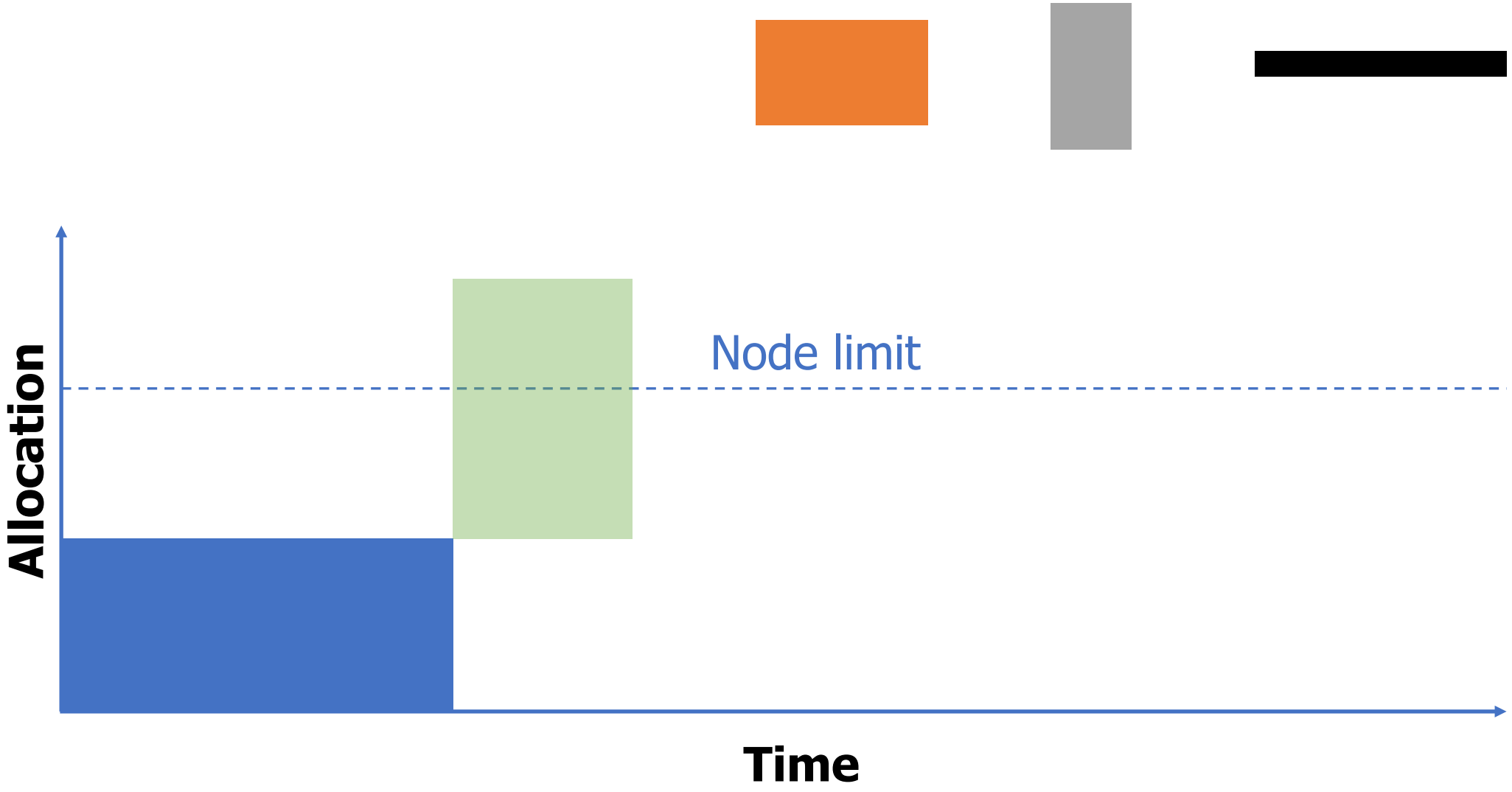
EASY-Shortest Job Backfill First (SJBF)



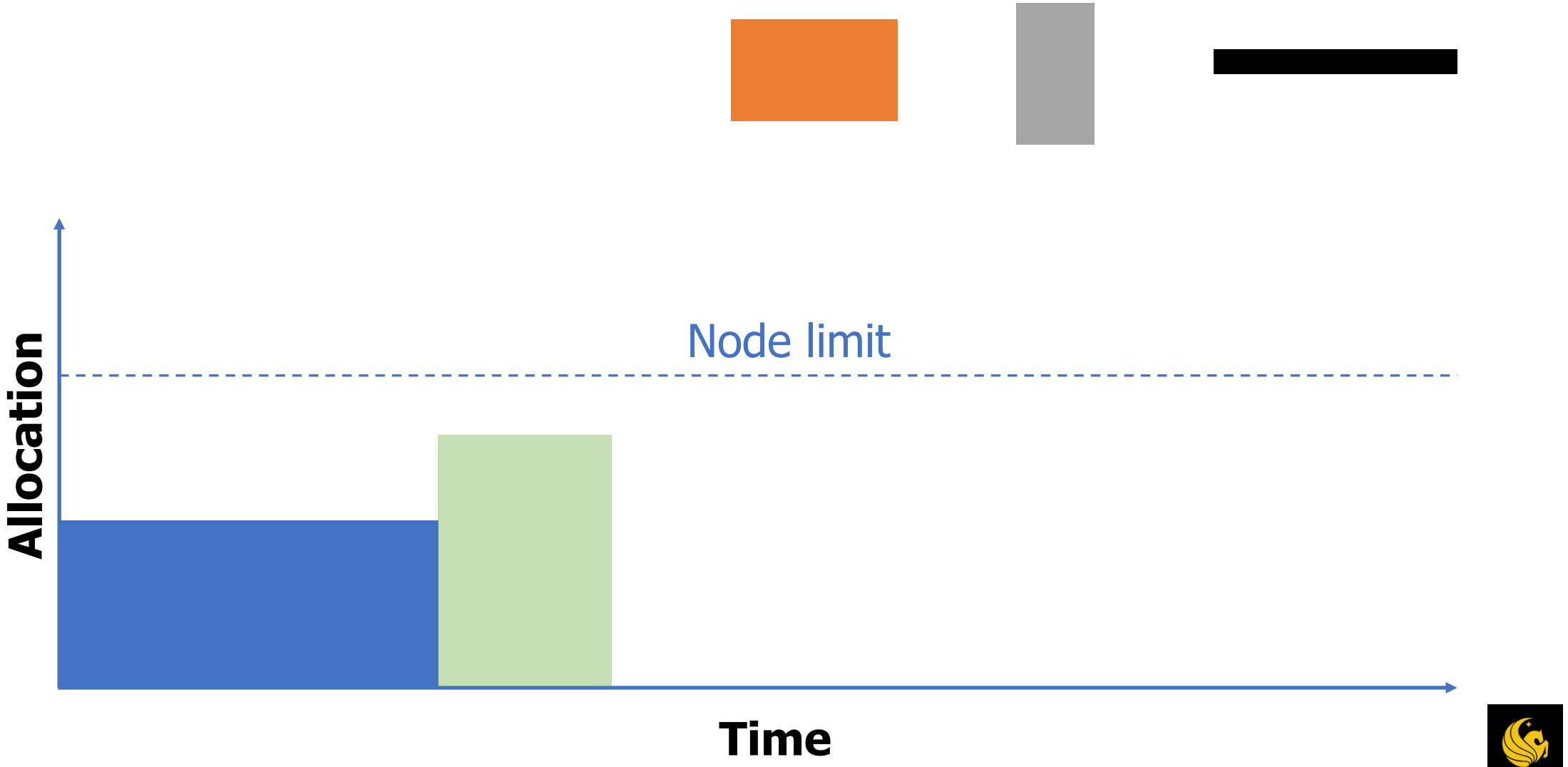
EASY-Shortest Job Backfill First (SJBF)



EASY-Shortest Job Backfill First (SJBF)

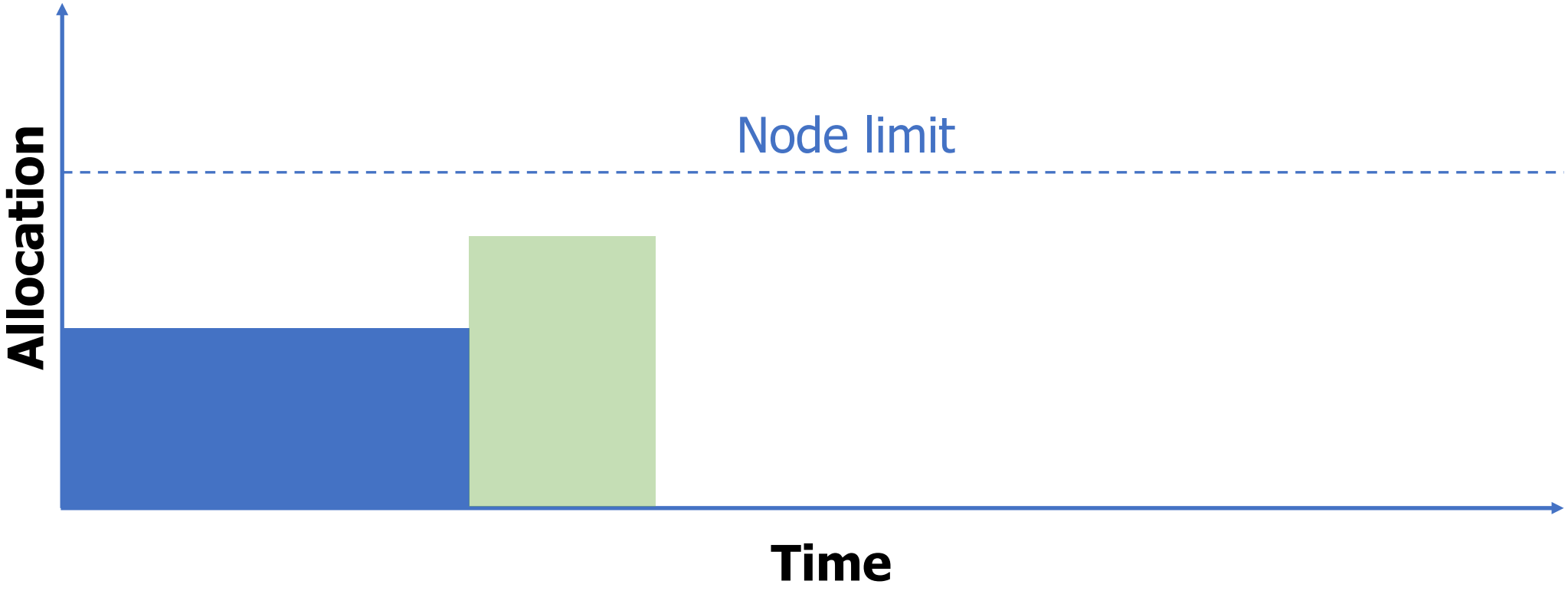


EASY-Shortest Job Backfill First (SJBF)



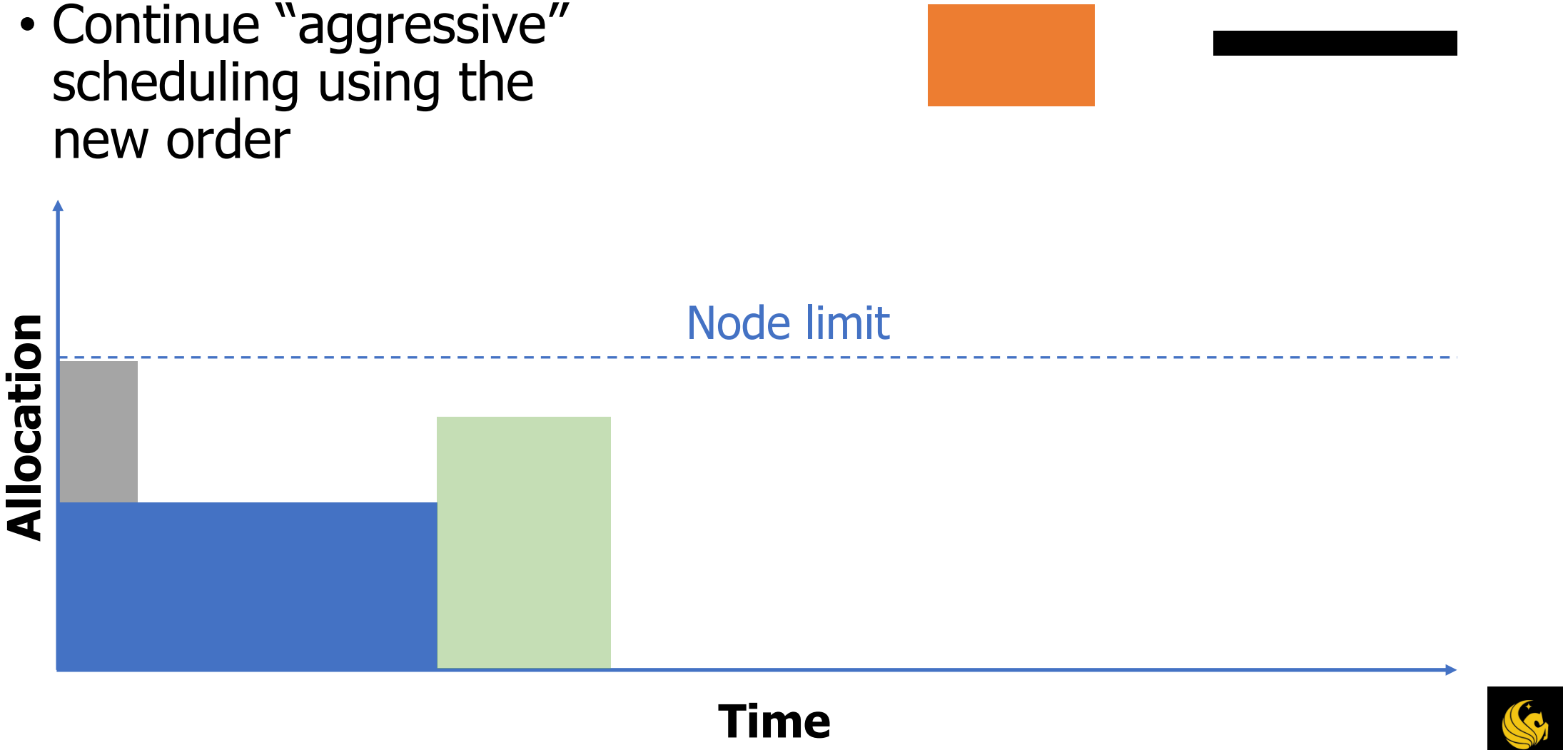
EASY-Shortest Job Backfill First (SJBF)

- Reorder jobs after the reservation



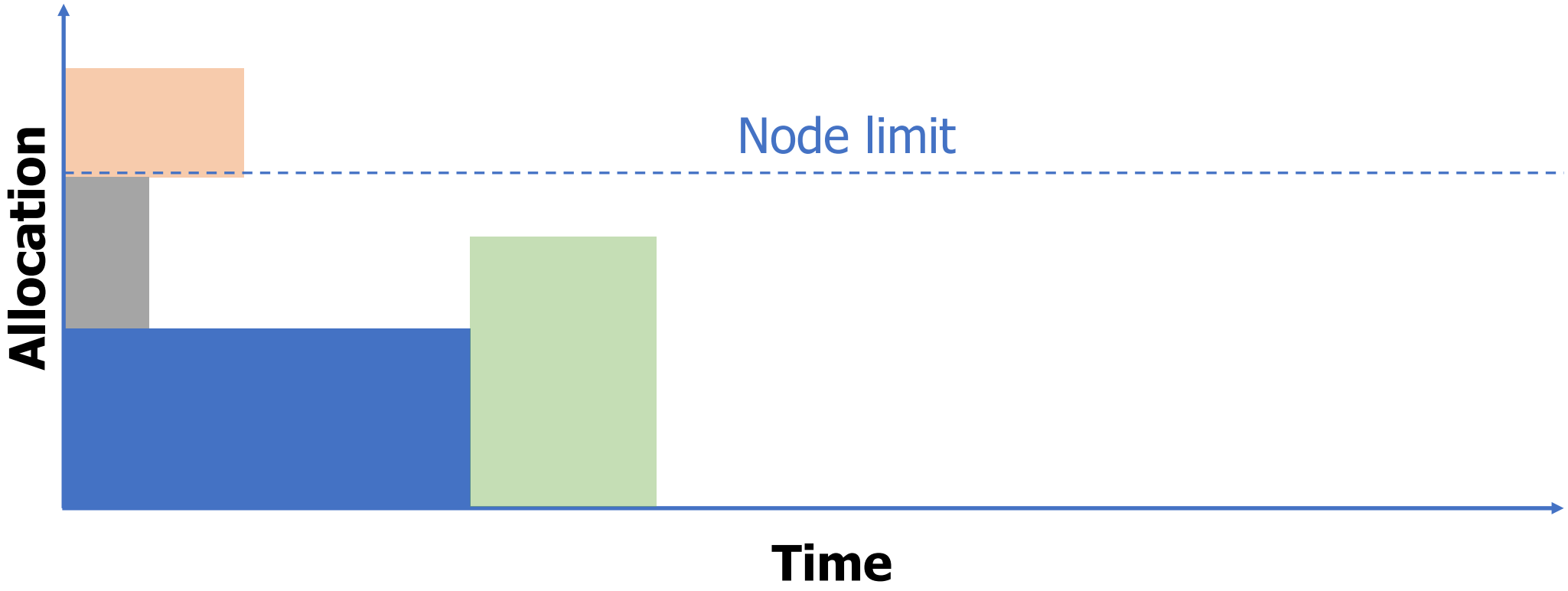
EASY-Shortest Job Backfill First (SJBF)

- Continue "aggressive" scheduling using the new order



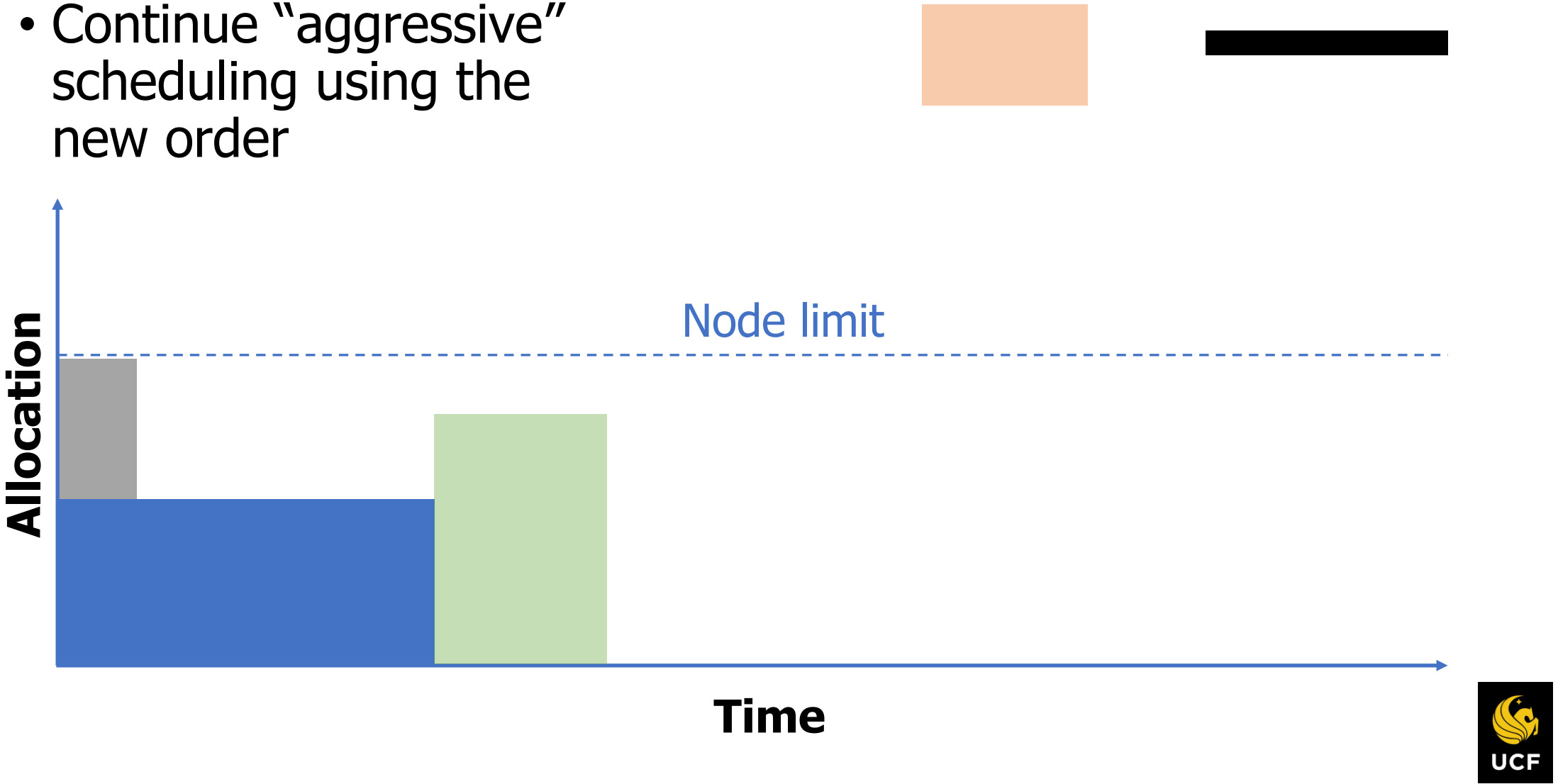
EASY-Shortest Job Backfill First (SJBF)

- Continue "aggressive" scheduling using the new order



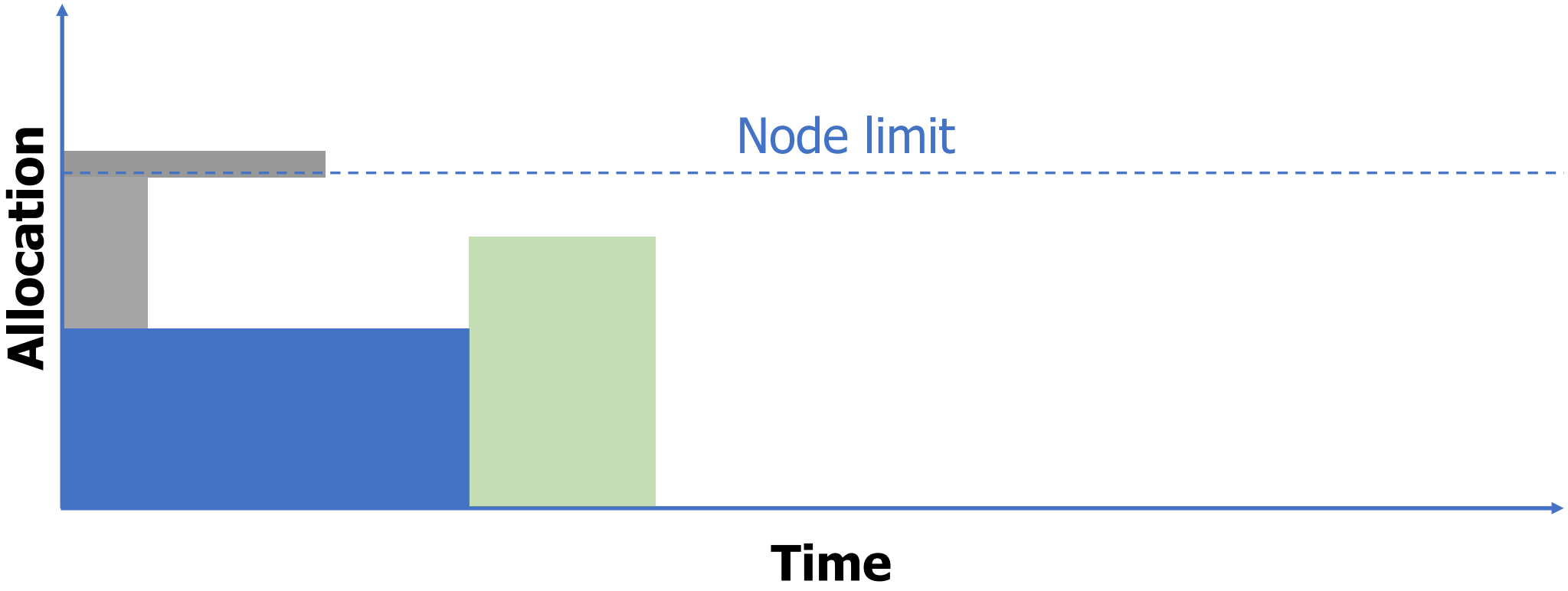
EASY-Shortest Job Backfill First (SJBF)

- Continue "aggressive" scheduling using the new order



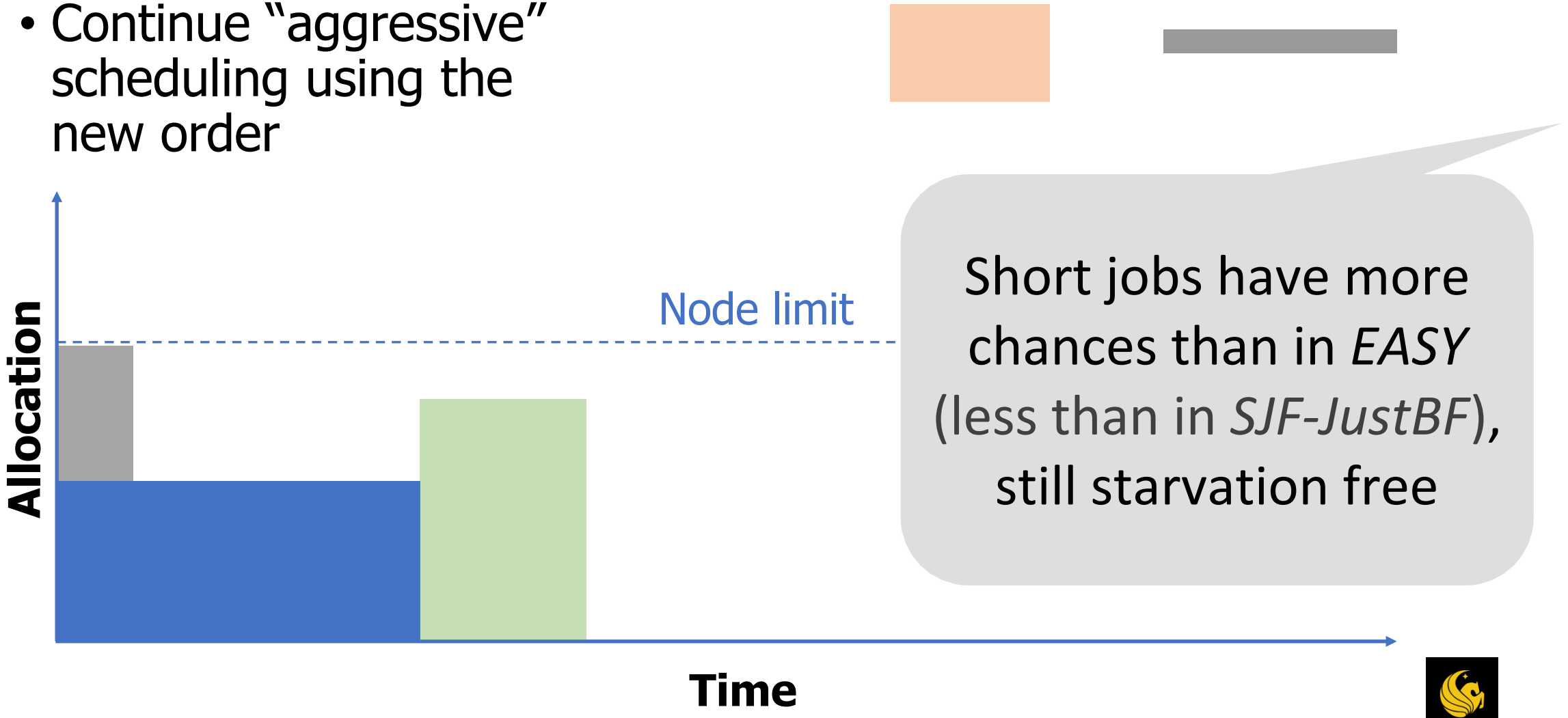
EASY-Shortest Job Backfill First (SJBF)

- Continue "aggressive" scheduling using the new order



EASY-Shortest Job Backfill First (SJBF)

- Continue "aggressive" scheduling using the new order



Short jobs have more chances than in *EASY* (less than in *SJF-JustBF*), still starvation free

Investigated algorithms

	JustBF	*EASY*
Default	*JustBF*	*EASY*
SJF	*SJF-JustBF*	*EASY-SJBF*
SAF	*SAF-JustBF*	
LAF	*LAF-JustBF*	

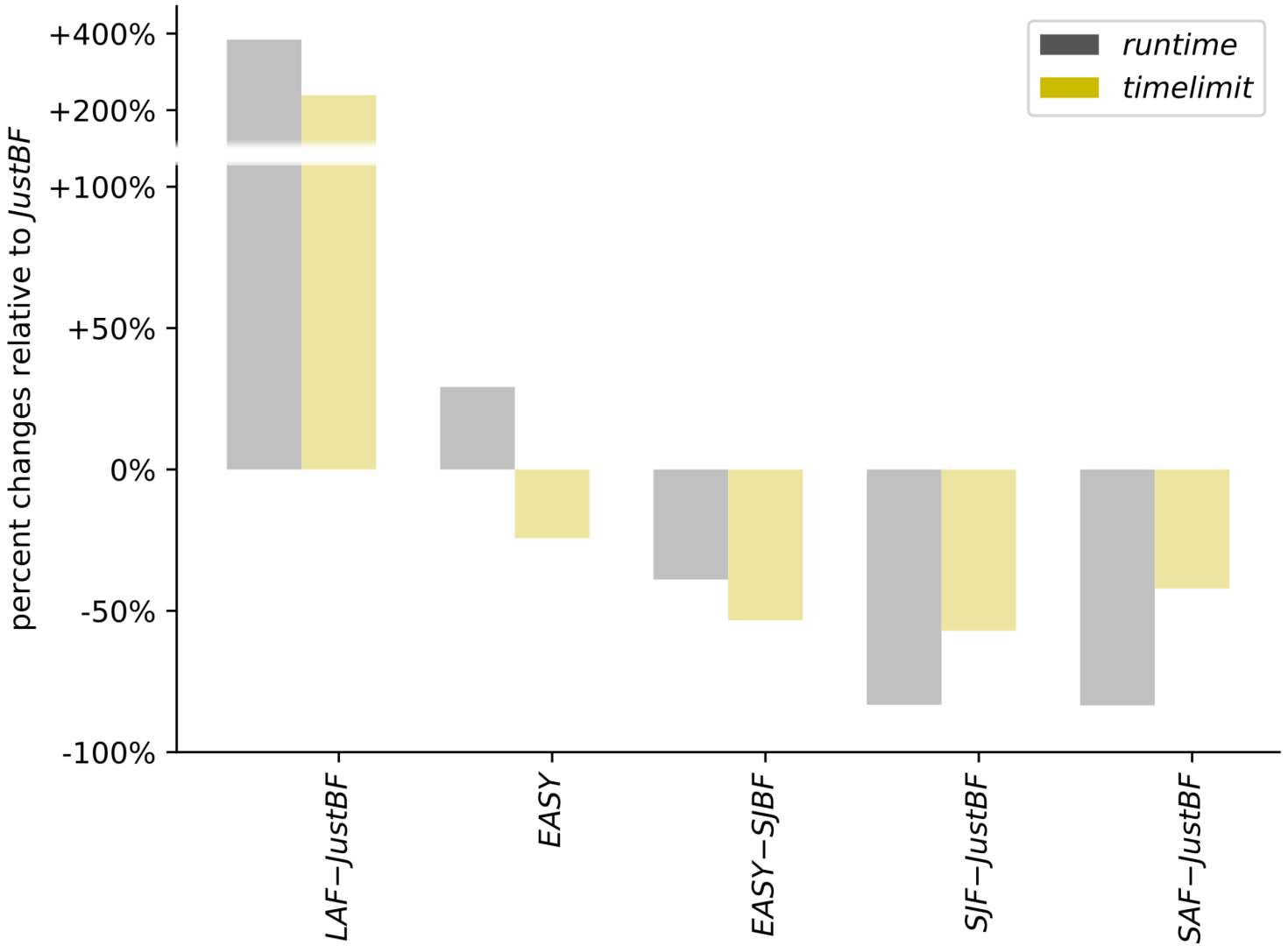
Workload logs used in benchmarking

	# CPU	# Jobs	Utilization	Year	Duration
KTH-SP2 [1]	100	28k	70%	1996	11 months
CTC-SP2 [1]	338	77k	85%	1996	11 months
SDSC-SP2 [1]	128	60k	83%	2000	24 months
SDSC-BLUE [1]	1,152	234k	77%	2003	32 months
CEA-CURIE [1]	93,312	313k	62%	2012	8 months
BW201911 [2]	22,636	92k	68%	2019	1 month

[1] http://www.cs.huji.ac.il/labs/

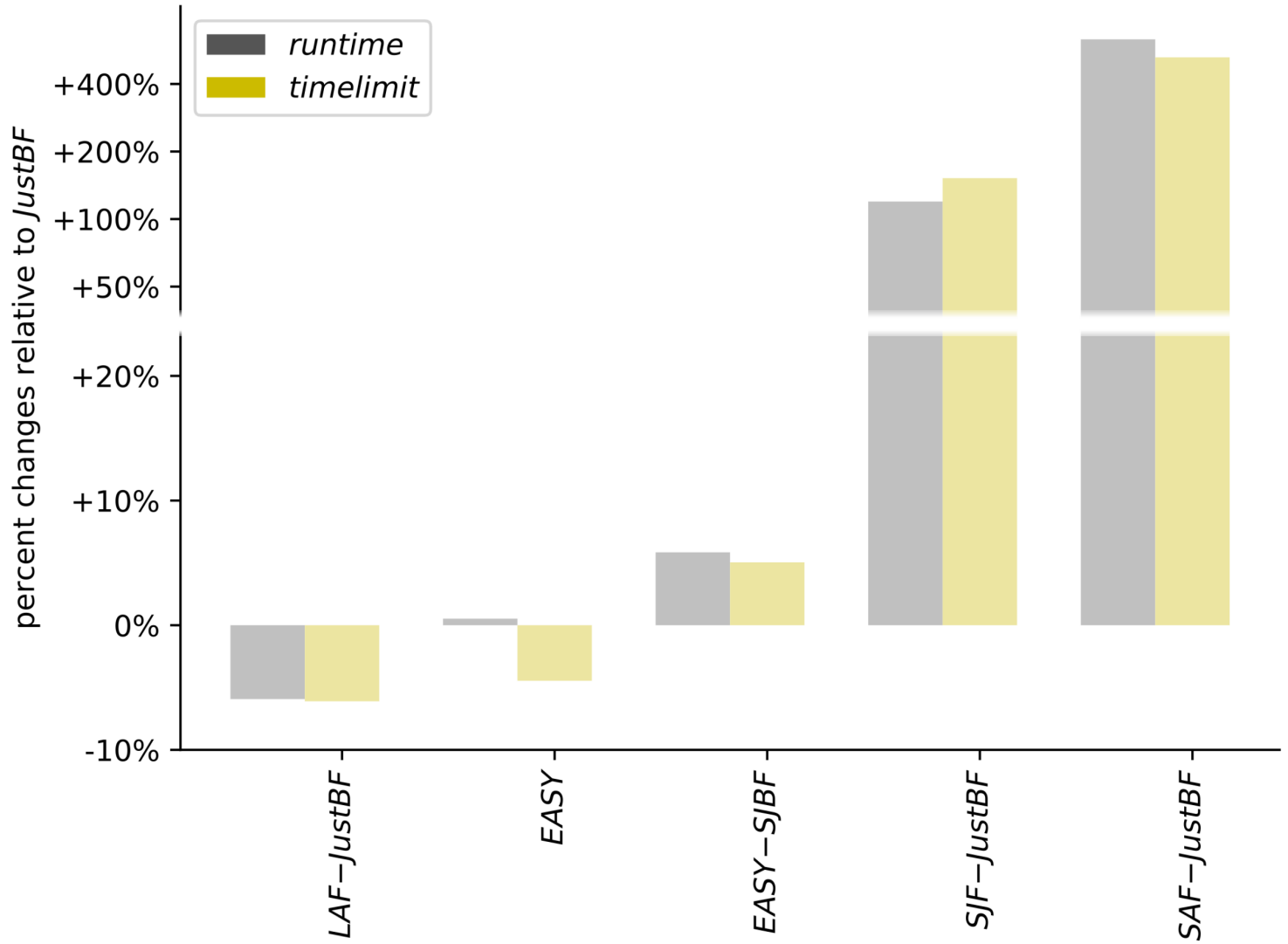
[2] https://bluewaters.ncsa.illinois.edu/data-sets

BSLD of schedules of *CTC-SP2*



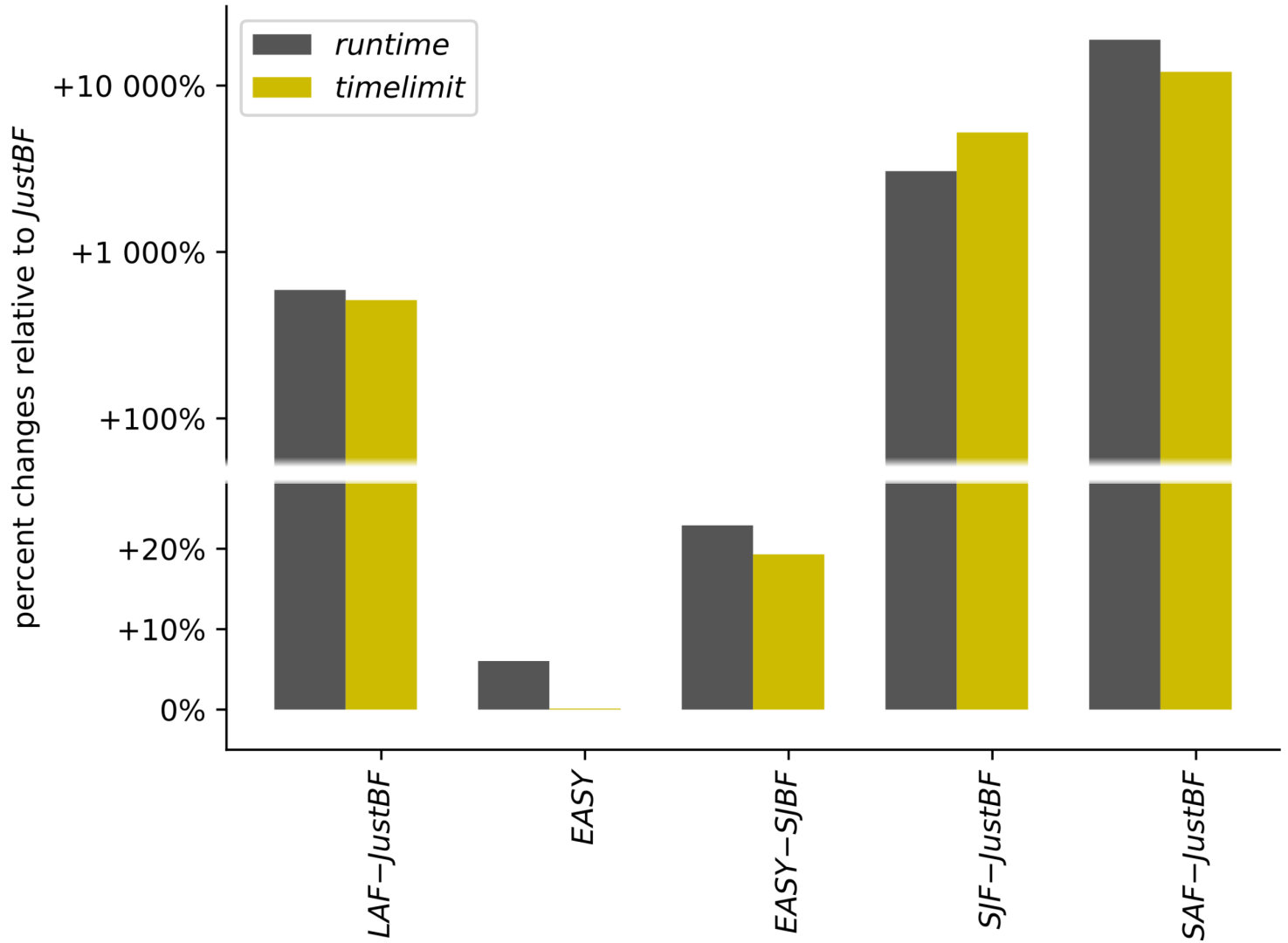
Lower is better

AWF of schedules of *CTC-SP2*



Lower is better

P^2SF of schedules of *CTC-SP2*



Lower is better

Scheduling Algorithms: Conclusions

- *JustBF* has best fairness and is best at P^2SF
- *SAF-Just* and *SJF-JustBF* are better at $BSLD$, at the expense of the packing efficiency and fairness
- *LAF-JustBF* is best in packing efficiency, at the expense of the other metrics
- When using timelimits, *EASY* is close to *JustBF* in packing efficiency and fairness

Example 2: Effect of Runtime Predictions on Scheduling Quality

Motivation

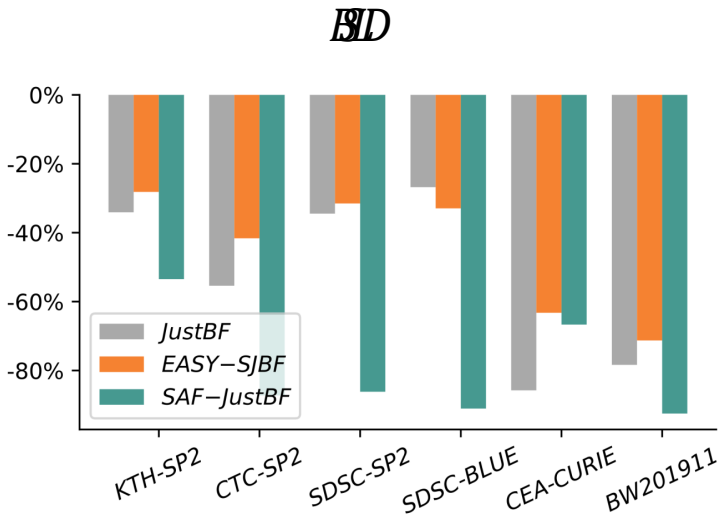
EASY–SJBF leads to better scheduling (according to $BSLD$) when using job runtime estimates (instead of timelimits)

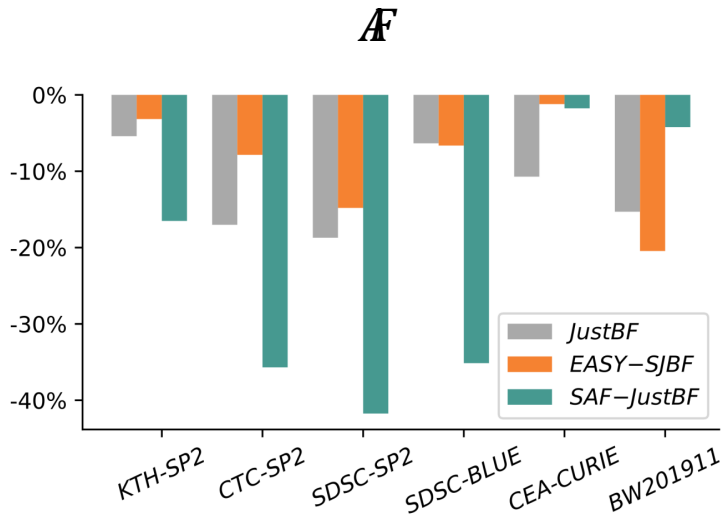
- Variants: "Last2", "CVH"

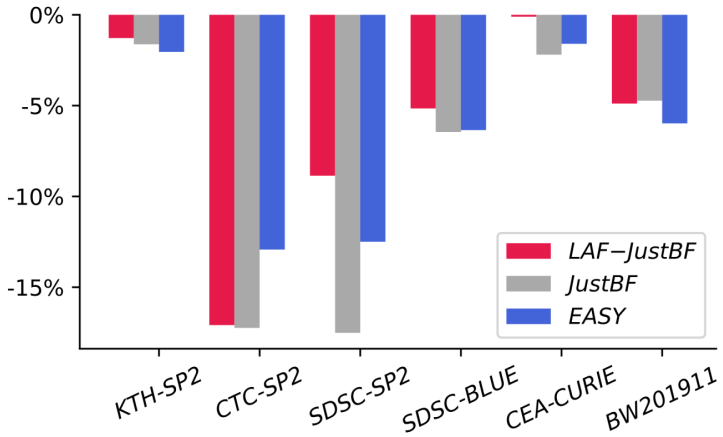
Plan

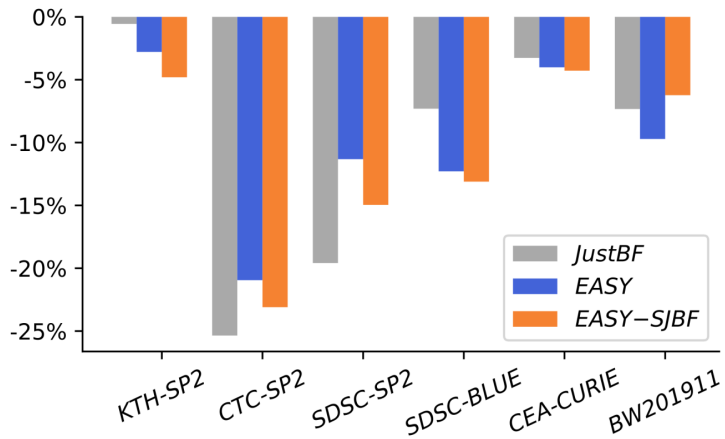
- Analyze how the estimates improve packing efficiency and fairness (when used with most suitable algorithms for these targets)
- Also analyze other possible predictors

Metrics improvement if job runtime is used instead of timelimit for scheduling









Last2 predictor

$$\left\langle \begin{matrix} predicted \\ running \\ time \end{matrix} \right\rangle = \begin{cases} \left\langle \begin{matrix} average\ runtime \\ of\ user's\ last\ 2\ jobs \end{matrix} \right\rangle, & if \begin{matrix} user\ submitted \\ 2+\ jobs \end{matrix} \\ \langle timelimit \rangle, & otherwise \end{cases}$$

D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, Jun. 2007, doi: 10.1109/TPDS.2007.70606.

CVT predictor "Cross-Validation Heuristic Triple"

Table 2: Features extracted from the *SWF* data, for job j, belonging to user k.

Feature	Meaning
$\widetilde{p_j}$	the time the user requested for her job.
$p_{j-1}^{(k)}$	the running time of the last job of the same user, or 0 if such a job does not exist.
$p_{j-2}^{(k)}$	the running time of the second-to-last job of the same user, or 0 if N/A.
$p_{j-3}^{(k)}$	the running time of the third-to-last job of the same user, or 0 if N/A.
$AVE_2^{(k)}(p)$	the average running time of the two last historically recorded jobs of the same user.
$AVE_3^{(k)}(p)$	the average running time of the three last historically recorded jobs of the same user.
$AVE_{all}^{(k)}(p)$	the average running time of all historically recorded jobs of the same user.
q_j	amount of (CPU) resource requested by job j.
$AVE_{hist,r_j}^{(k)}(q)$	average historical resource request of user k, taken at release date of job j.
$\frac{q_j}{AVE_{hist,r_j}^{(k)}(q)}$	amount of resource requested normalized by average resource request.
$AVE_{curr,r_j}^{(k)}(q)$	average resource request of the user's currently running jobs, at release date
Jobs Currently Running	number of jobs of the user running, at release date
Longest Current running time	longest running time (so-far) of the user's currently running jobs, at release date
Sum Current running times	sum of the running times (so-far) of the user's currently running jobs, at release date
Occupied Resources	total size of resources currently being allocated to the same user.
Break Time	time elapsed since last job completion from the same user.
$\begin{cases} cos(\frac{2*\pi}{t_{day}} * (r_j \mod t_{day})) \\ sin(\frac{2*\pi}{t_{day}} * (r_j \mod t_{day})) \end{cases}$	time of the day the job was released. The periodic feature is decomposed into its cosinus and sinus, using the day period t_{day} (length of a day in seconds)
$\begin{cases} cos(\frac{2*\pi}{t_{week}} * (r_j \mod t_{week})) \\ sin(\frac{2*\pi}{t_{week}} * (r_j \mod t_{week})) \end{cases}$	time of the week the job was released. The periodic feature is decomposed into its cosinus and sinus, using the week period t_{week} (length of a day in seconds)

$$\mathfrak{L}(\mathbf{x}_j, f(\mathbf{x}_j), p_j) = \begin{cases} \gamma_j . L_u(f(\mathbf{x}_j) - p_j) & \text{if } f(\mathbf{x}_j) \geq p_j \\ \gamma_j . L_o(p_j - f(\mathbf{x}_j)) & \text{if } f(\mathbf{x}_j) < p_j \end{cases}$$

Table 3: Weighting factors considered for training the model. The constants are chosen to ensure positivity of the weights with typical running times and resource requests in the HPC domain. Logarithms are used to alleviate the high range produced by ratios.

γ_j	Interpretation
1	Constant weight.
$5 + \log(\frac{q_j}{p_j})$	Short jobs with large resource request should be well-predicted.
$5 + \log(\frac{p_j}{q_j})$	Long jobs with small resource request should be well-predicted.
$11 + \log(\frac{1}{q_j . p_j})$	Jobs of small "area" should be well-predicted.
$\log(q_j . p_j)$	Jobs of large "area" should be well-predicted.

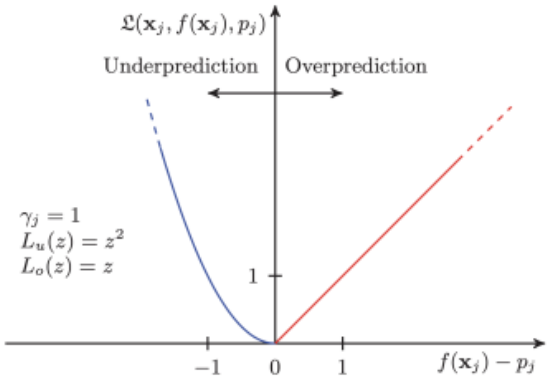


Figure 1: Example Loss function $\mathfrak{L}$, plotted with respect to the difference of it's second and third parameters $f(\mathbf{x}_j) - p_j$ (the prediction error).

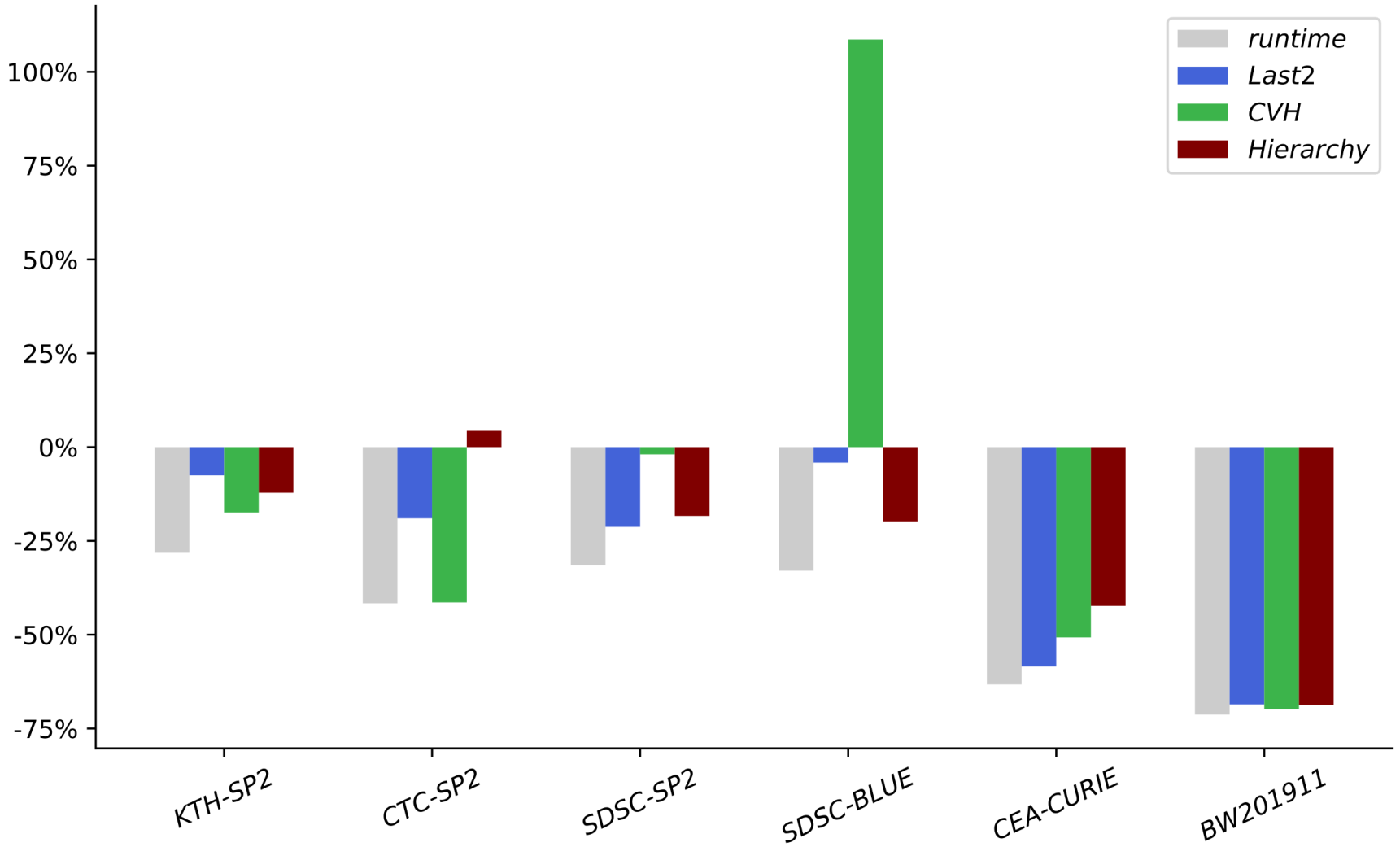
E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2015, pp. 1–10. doi: 10.1145/2807591.2807646.

Hierarchy predictor "Hierarchy of templates"

job/user/time/node
job/user/time/
job/user//node
job/user//
job//time/node
job//time/
job///node
job///
/user/time/node
/user/time/
/user//node
/user//
//time/node
//time/
///node
///

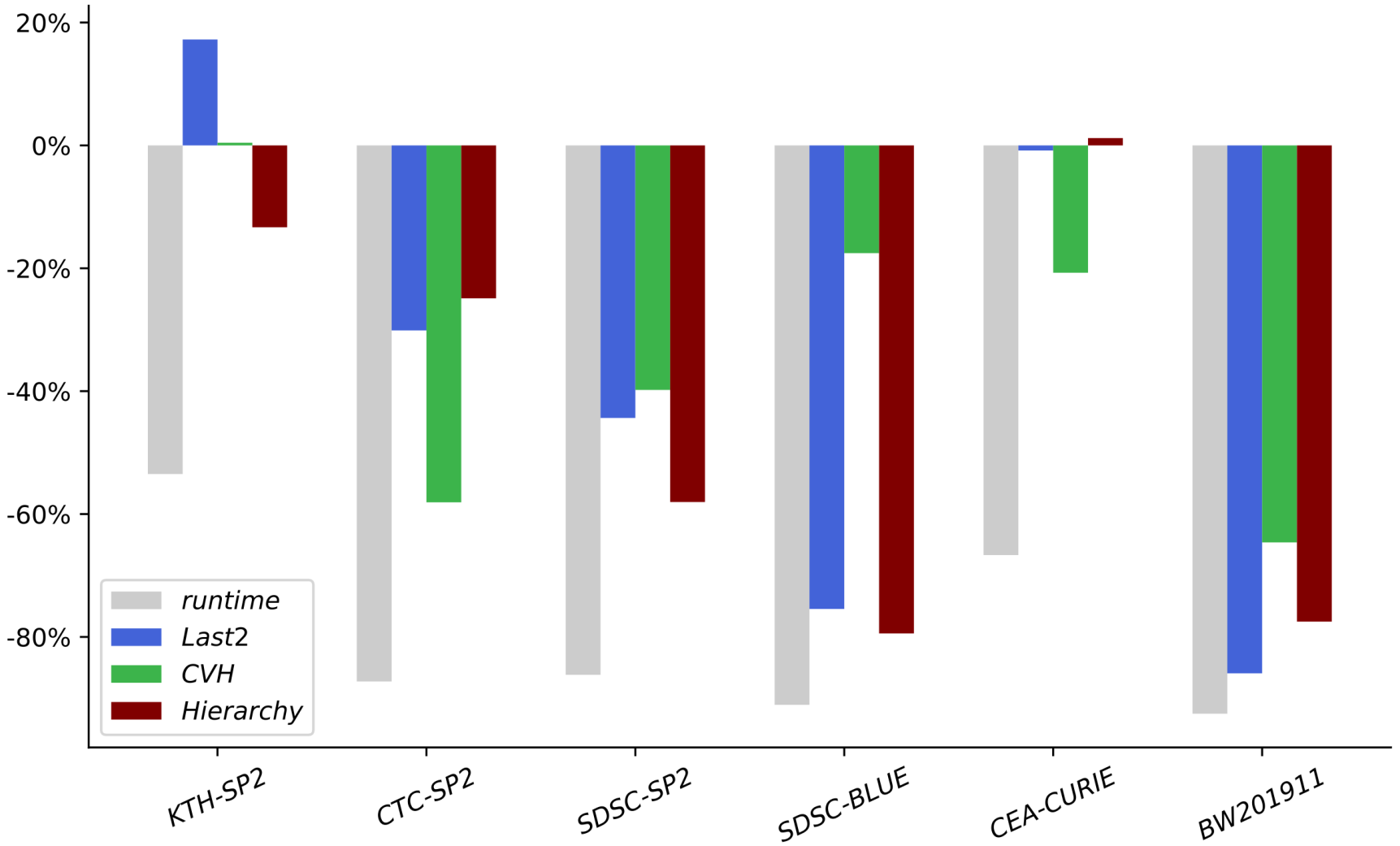
- Jobs are classified according to parameters known at submission
  - user id
  - job name or executable id
  - timelimit provided by user
  - required number of nodes
- Resource requirement is predicted from resource utilization of previous jobs from same group
  - Exponentially decaying average
- If no jobs has finished for the group, a more general template from the hierarchy is tried
- When a job finishes, predictions of all 16 groups are updated

BSLD of *EASY-SJBF* relative change
when replacing timelimit with runtime estimates



Lower is better

BSLD of *SAF-JustBF* relative change
when replacing timelimit with runtime estimates

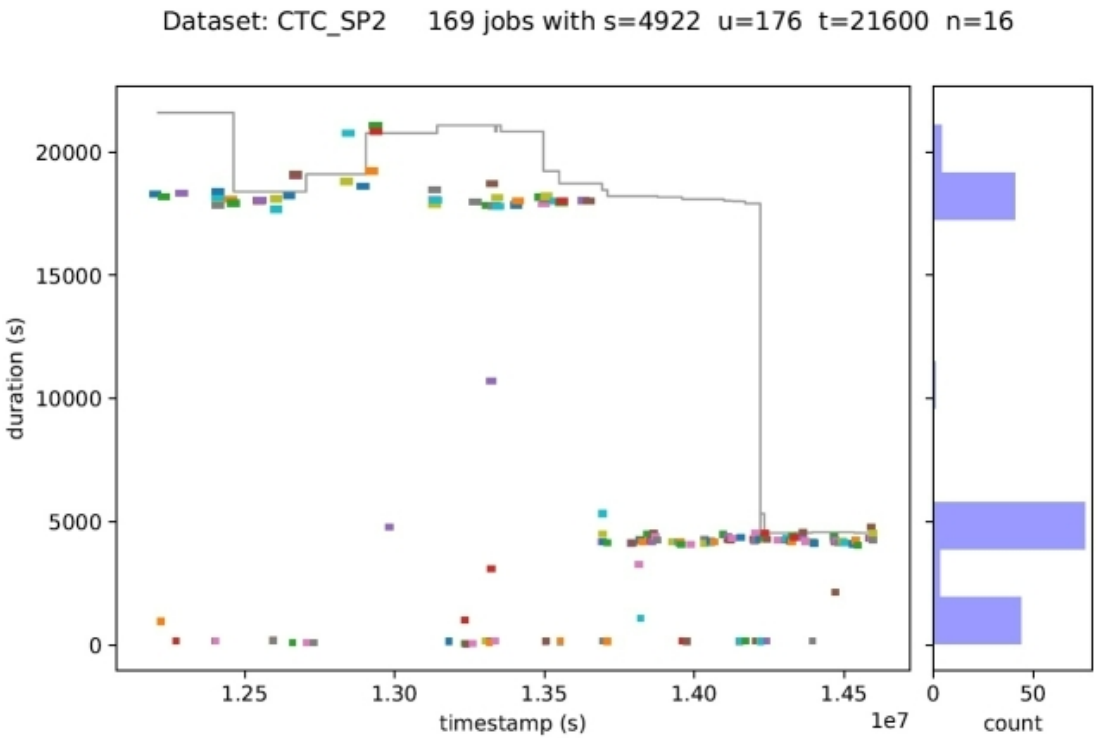


Lower is better

SP-x predictor
"Survival probability x percent"

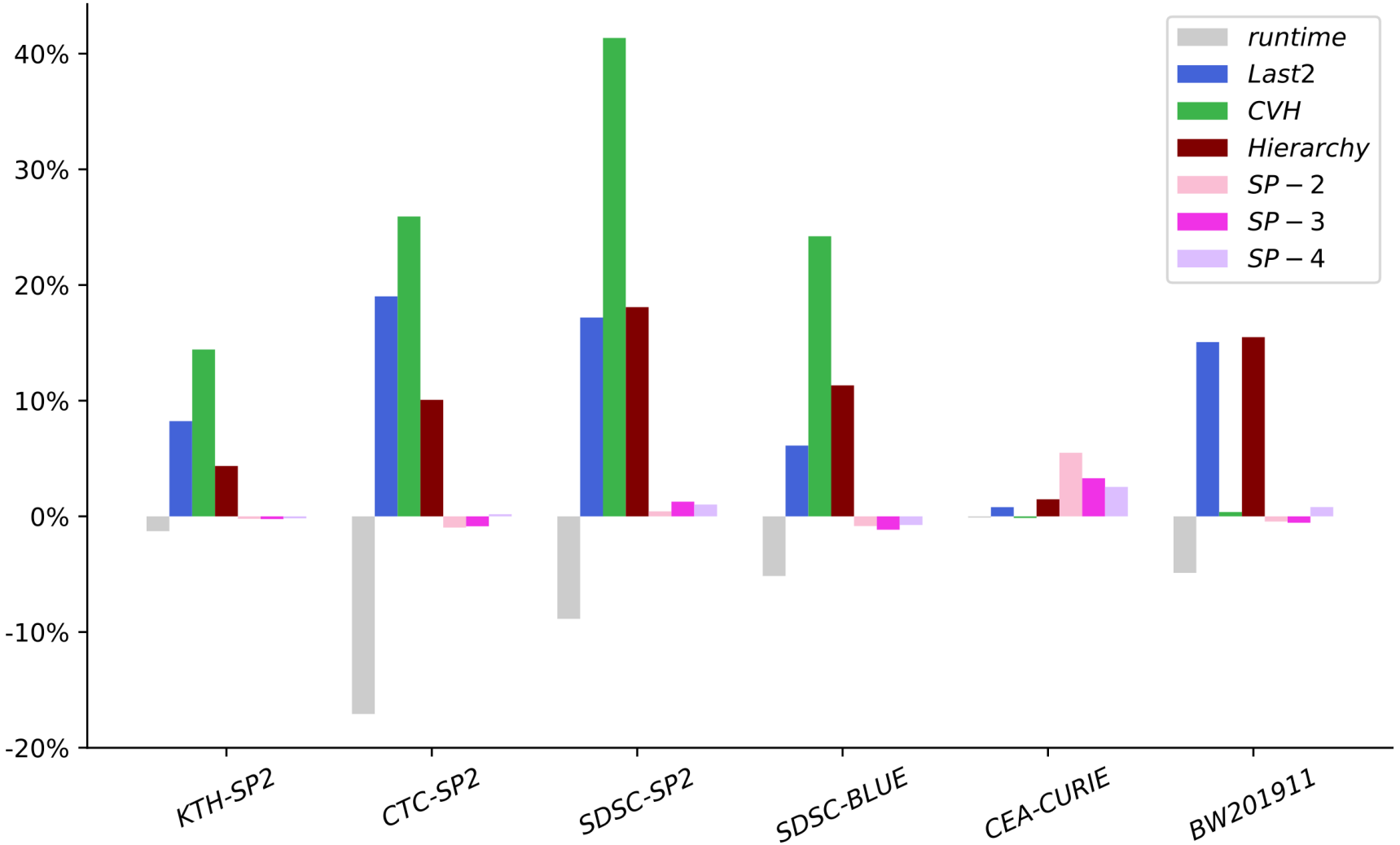
- Group jobs using all parameters (job/user/time/node)
- Tracks a threshold that separates less than x % of longest jobs from the rest
  - $x = 2, 3,$ or 4
  - exponentially decaying weights
- Predicts along that threshold

➢ *Less underprediction in the presence of multi-modal distributions*



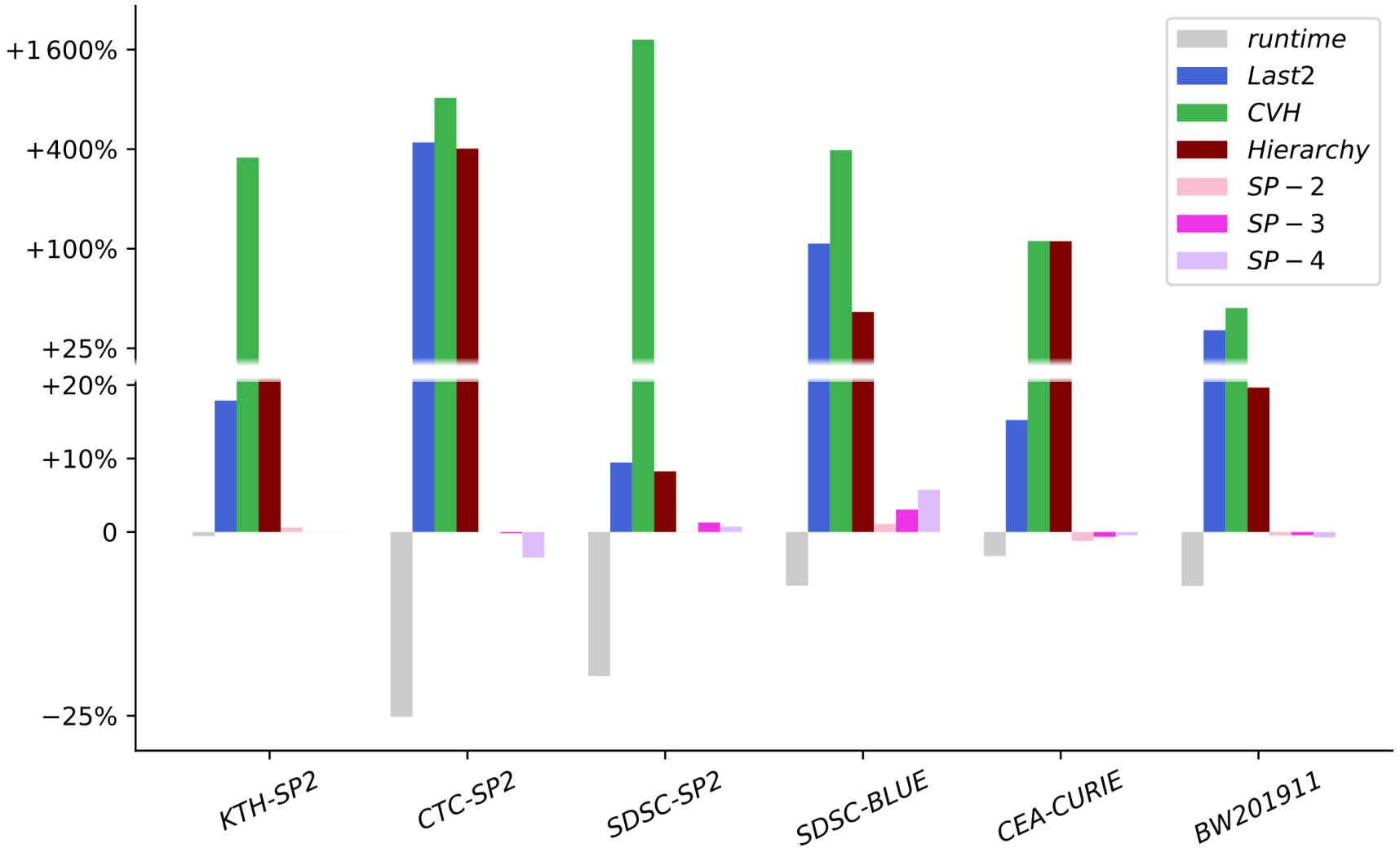
K. Lamar, A. Goponenko, C. Peterson, B. A. Allan, J. M. Brandt, and D. Dechev, "Backfilling HPC Jobs with a Multimodal-Aware Predictor," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2021, pp. 618–622. doi: 10.1109/Cluster48925.2021.00093.

AWF of *LAF-JustBF* relative change
when replacing timelimit with runtime estimates



Lower is better

P^2SF of *JustBF* relative change
when replacing timelimit with runtime estimates



Lower is better

Runtime Predictions: Conclusions

- Improving $BSLD$ is easy
- Improving AWF and $P^{\propto}SF$ by using predictions of running time is difficult
  - Underprediction hurts packing efficiency and fairness more than overprediction
- Current state of the art in estimating job runtime is not adequate for practical application in job scheduling

Conclusions

- Schedule quality metrics for-
  - Packing efficiency (AWF)
  - Both packing efficiency and compliance with priorities ($P^{\propto}SF$)
- Proof-of-concept examples (comparing scheduling algorithms and evaluating runtime prediction approaches)
  - Illustration of trade-offs between improving $BSLD$ (or AF) and preserving job packing efficiency and fairness
- Improving $JustBF$'s packing efficiency and fairness is hard
- Better methods of estimating job runtime are needed
  - Underprediction hurts packing efficiency and fairness more than

Acknowledgements

- Thanks to Benjamin Schwaller, Omar Aaziz, Kevin Stroup, and Cory Lueninghoener for valuable discussions and help throughout the project
- Thanks to Prof. Dechev team at UCF
- The works at the University of Central Florida were supported through contracts with Sandia National Laboratories

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND #: TODO

UCF

Contacts

agoponenko@knights.ucf.edu
(Alexander Goponenko)

kenneth@knights.ucf.edu
(Kenneth Lamar)

clp8199@knights.ucf.edu
(Christina Peterson)

baallan@sandia.gov
(Benjamin Allan)

brandt@sandia.gov
(Jim Brandt)

Damian.Dechev@ucf.edu
(Damian Dechev)