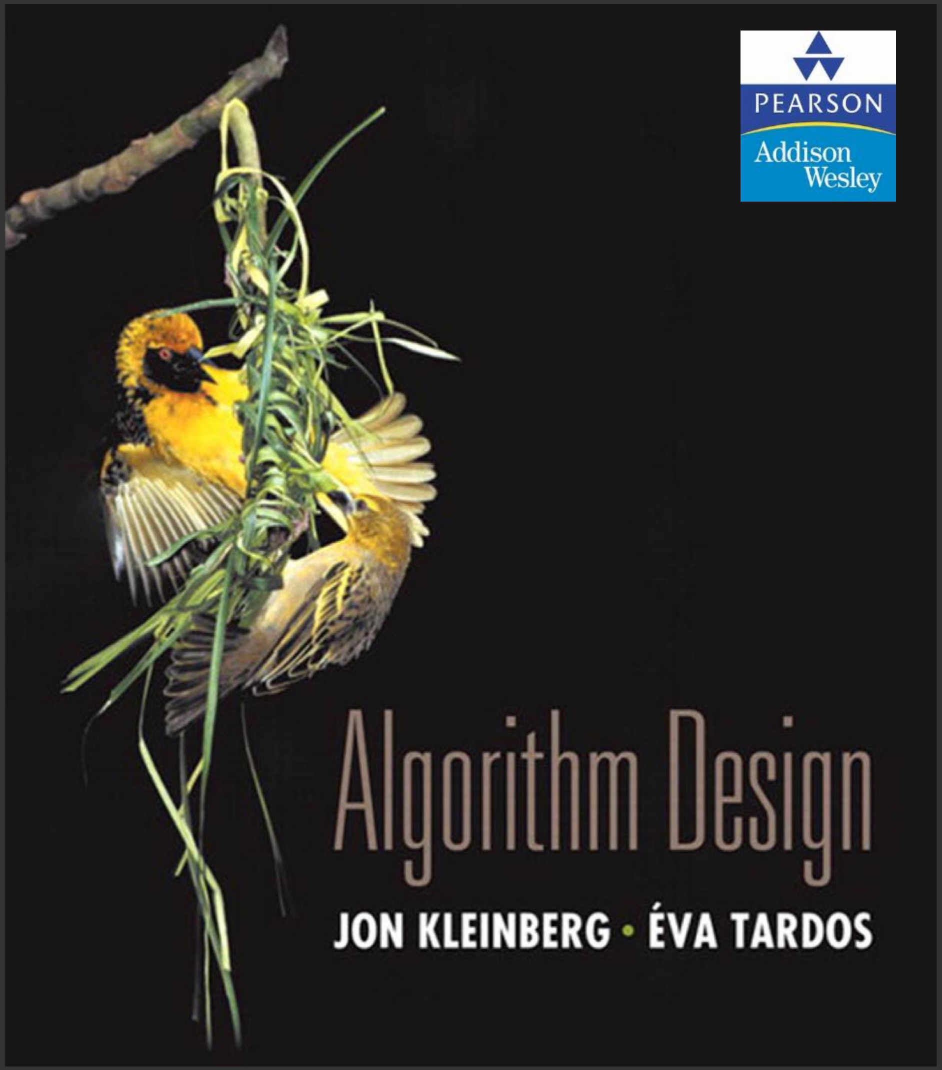


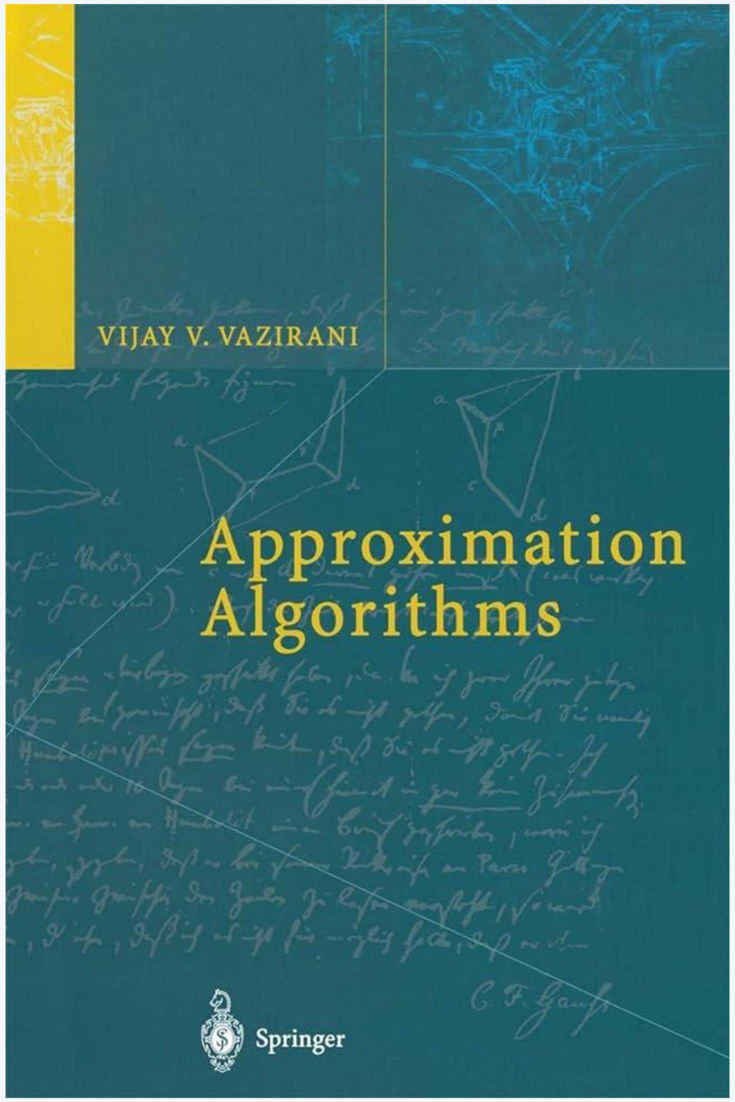
11. APPROXIMATION ALGORITHMS

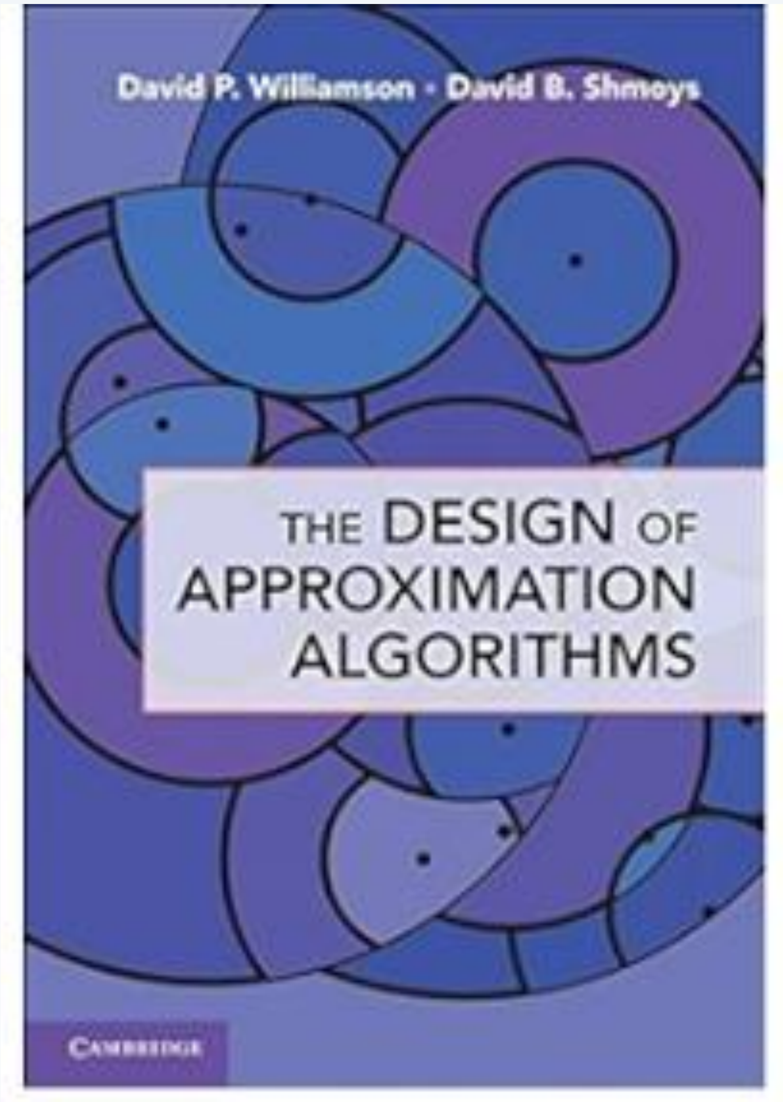
- *load balancing*
- *center selection*

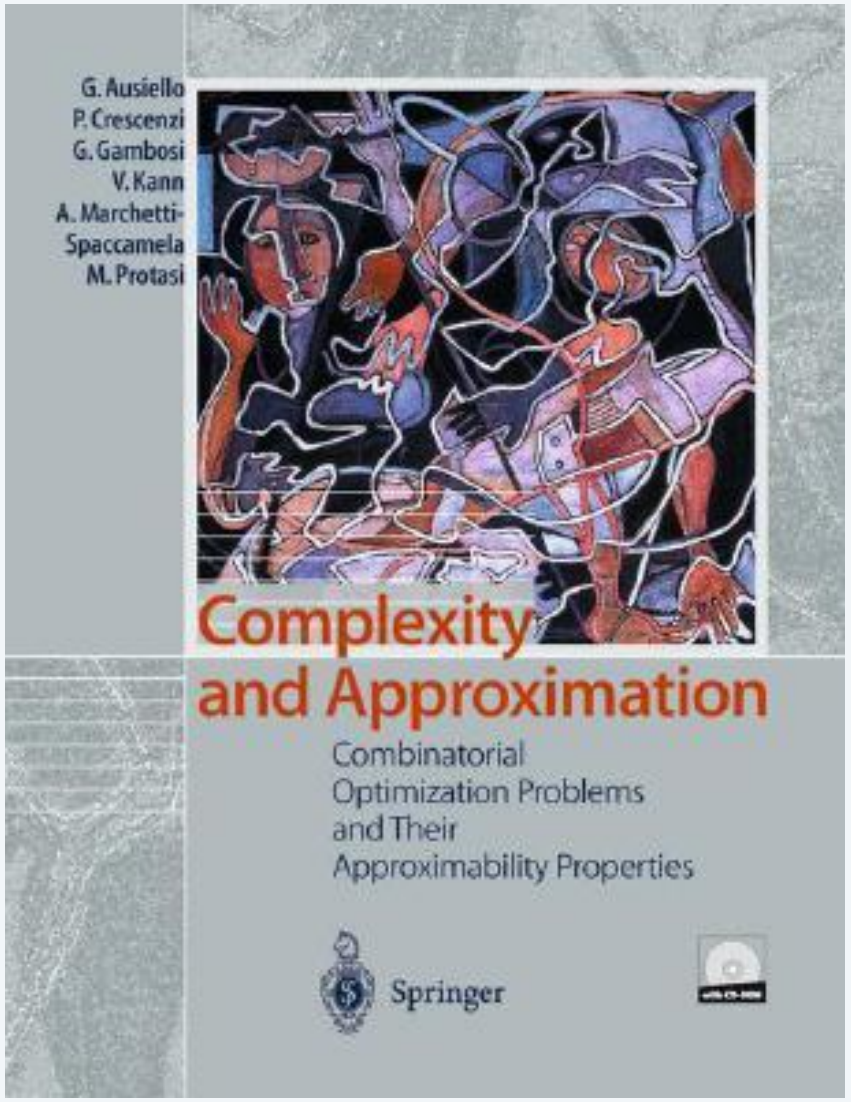
Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Approximation algorithms: well-established field







Coping with NP-completeness

Q. Suppose I need to solve an **NP**-hard optimization problem. What should I do?

A. Sacrifice one of three desired features.

i. Runs in polynomial time.
ii. Solves arbitrary instances of the problem.
iii. Finds optimal solution to problem.

ρ-approximation algorithm.

- Runs in polynomial time.
- Solves arbitrary instances of the problem
- Finds solution that is within ratio ρ of optimum.

Challenge. Need to prove a solution's value is close to optimum, without even knowing what is optimum value.

Def.
An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α the value of an optimal solution.

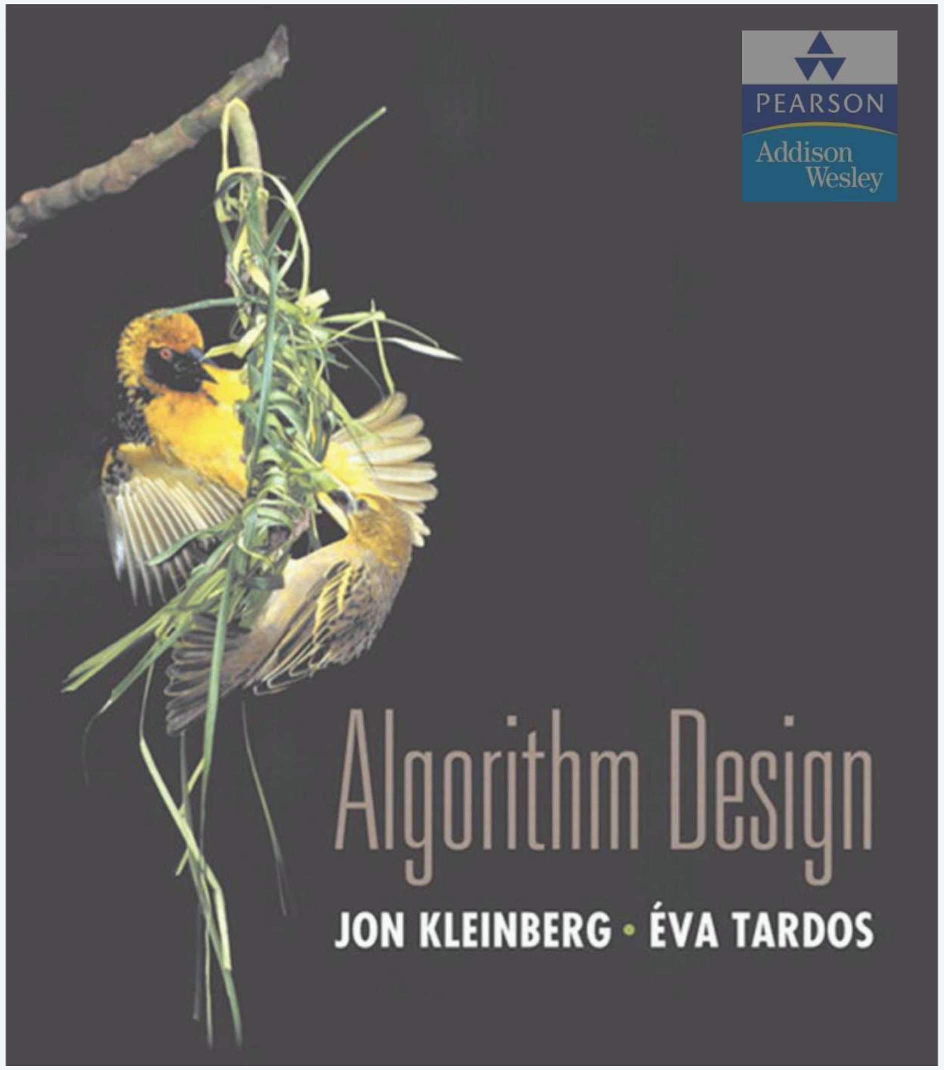
α: approximation ratio or approximation factor

minimization problem:

- $\alpha \geq 1$
- for each returned solution x, $\text{cost}(x) \leq \alpha \, \text{OPT}(x)$

maximization problem:

- $\alpha \leq 1$
- for each returned solution x, $\text{value}(x) \geq \alpha \, \text{OPT}(x)$



SECTION 11.1

11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*

Load balancing

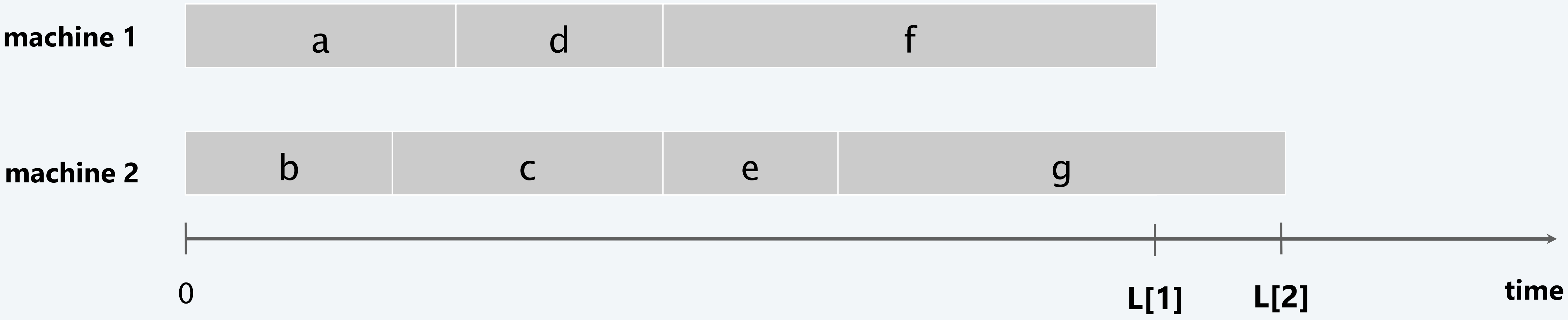
Input. m identical machines; $n \geq m$ jobs, job j has processing time t_j.

- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let $S[i]$ be the subset of jobs assigned to machine i.
The load of machine i is $L[i] = \Sigma_{j \in S[i]} t_j$.

Def. The makespan is the maximum load on any machine $L = \max_i L[i]$.

Load balancing. Assign each job to a machine to minimize makespan.

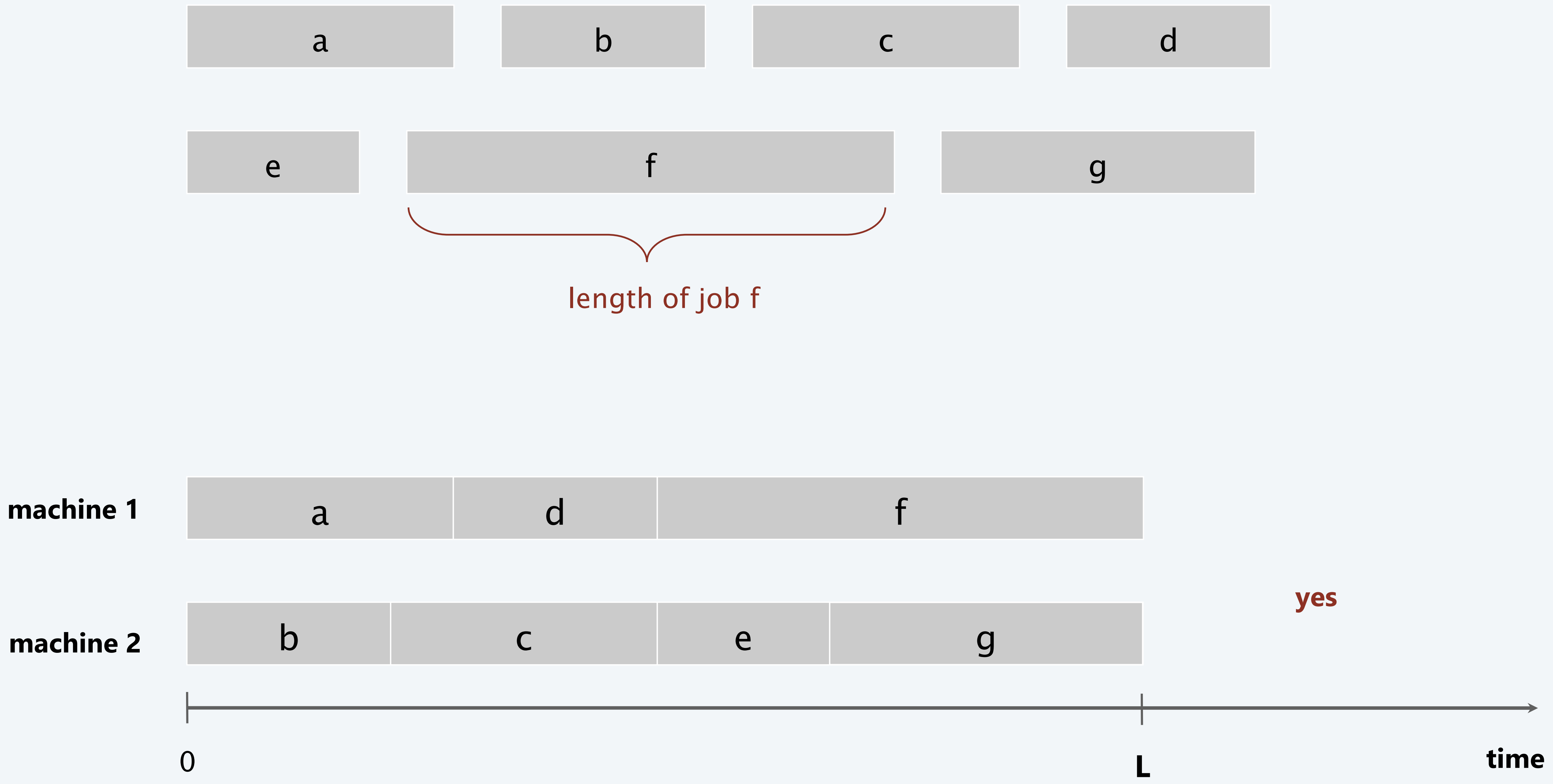


Load balancing on 2 machines is NP-hard

Claim. Load balancing is hard even if $m = 2$ machines.

Pf. PARTITION $\leq_P$ LOAD-BALANCE.

NP-complete by Exercise 8.26



length of job f

yes

Load balancing: list scheduling

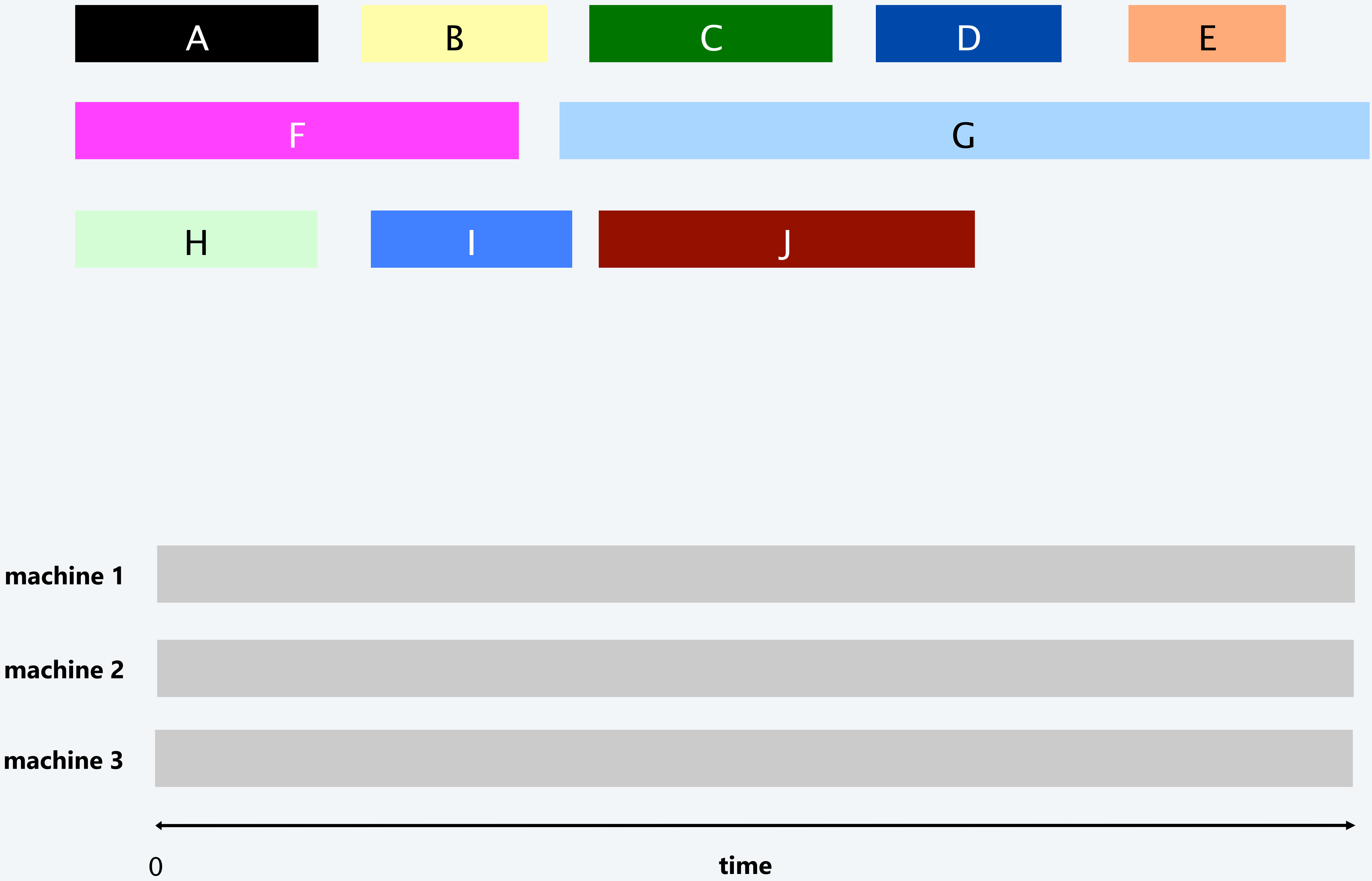
List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign job j to machine i whose load is smallest so far.

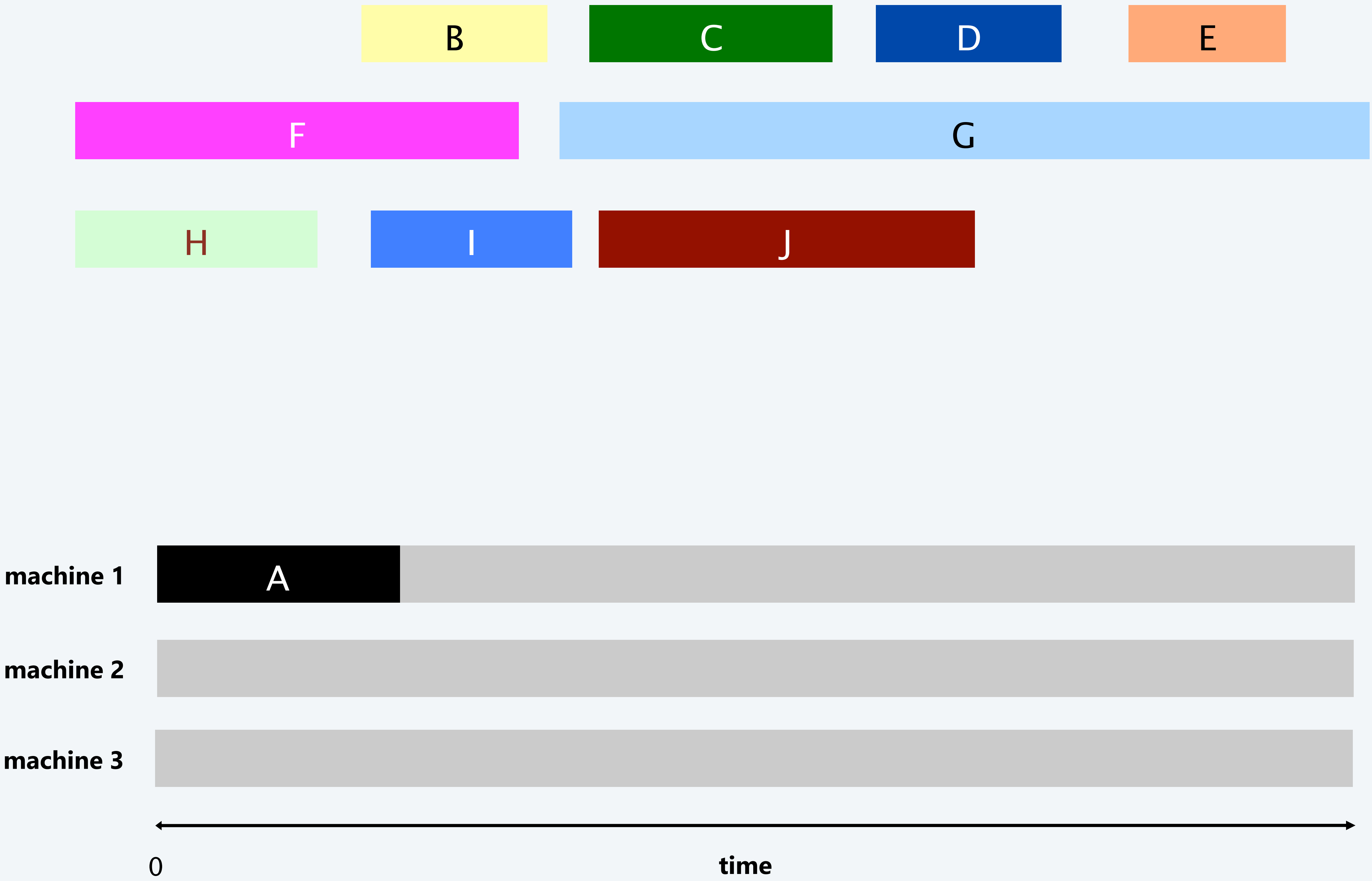
```
LIST-SCHEDULING $(m, n, t_1, t_2, \ldots, t_n)$

FOR $i = 1$ TO $m$
    $L[i] \leftarrow 0.$                        ← load on machine $i$
    $S[i] \leftarrow \varnothing.$              ← jobs assigned to machine $i$

FOR $j = 1$ TO $n$
    $i \leftarrow \text{argmin}_k \, L[k].$     ← machine $i$ has smallest load
    $S[i] \leftarrow S[i] \cup \{ j \}.$        ← assign job $j$ to machine $i$
    $L[i] \leftarrow L[i] + t_j.$               ← update load of machine $i$

RETURN $S[1], S[2], \ldots, S[m].$
```

Implementation. $O(n \log m)$ using a priority queue for loads $L[k]$.

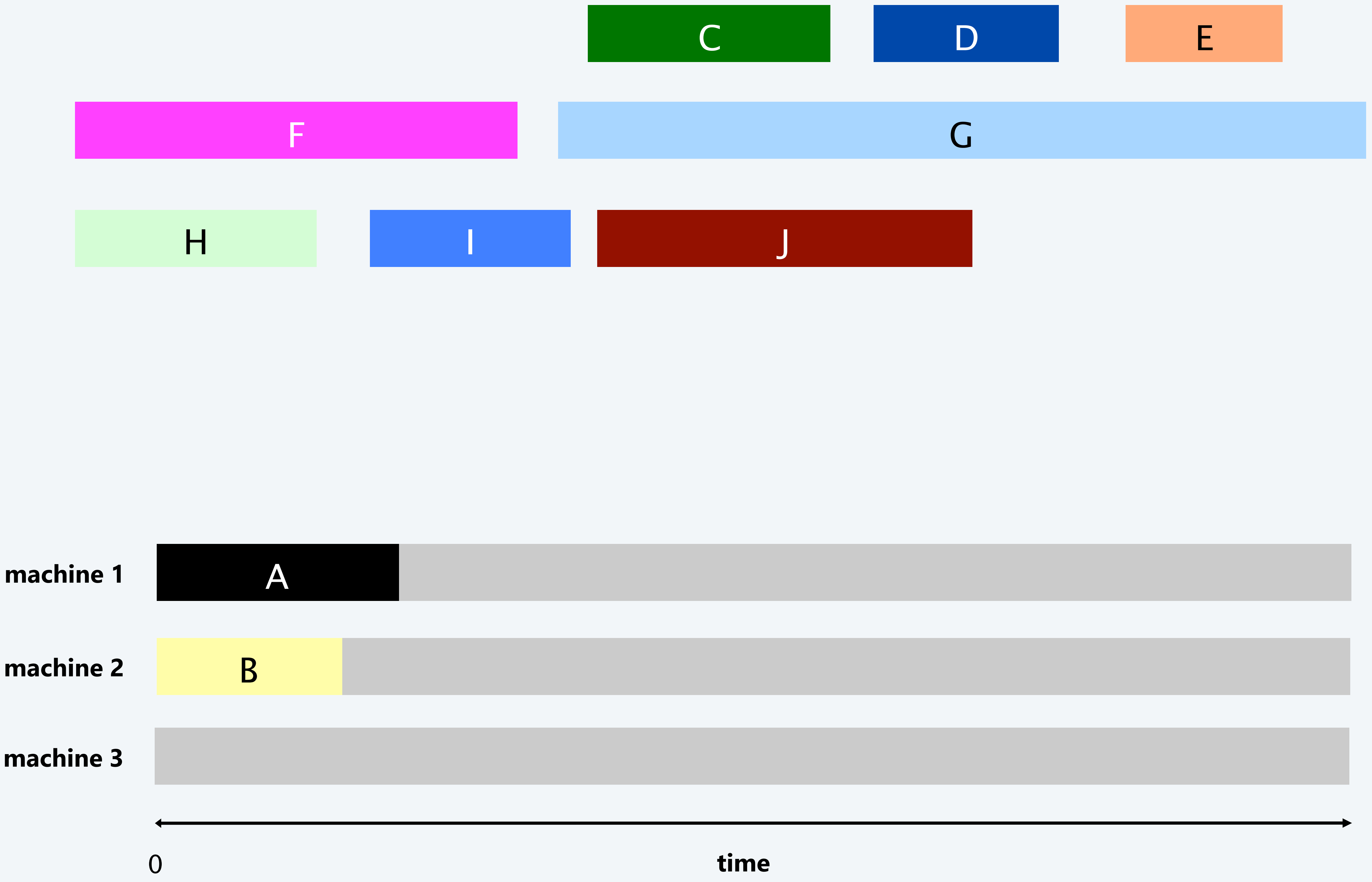
List scheduling demo



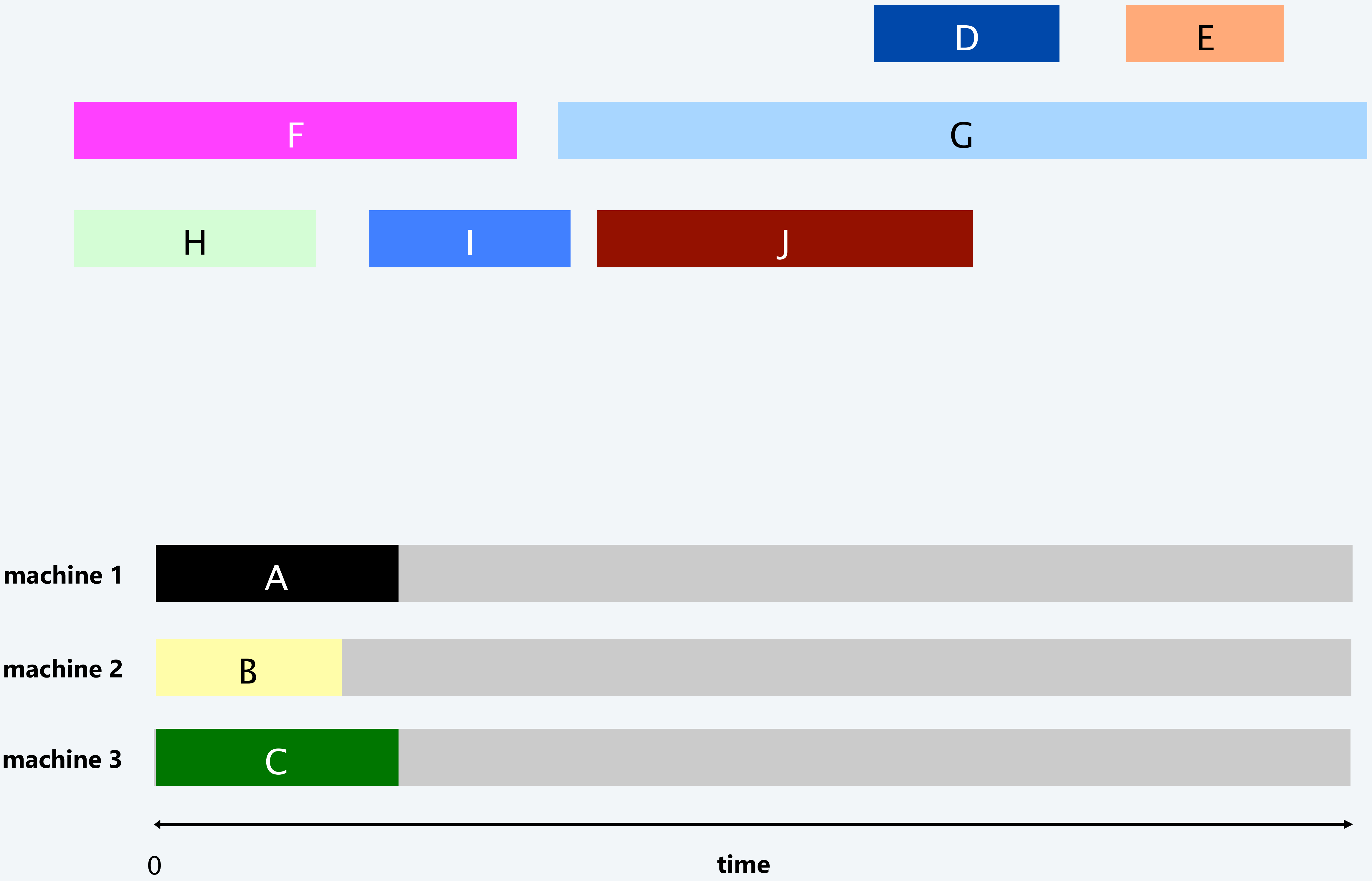
List scheduling demo



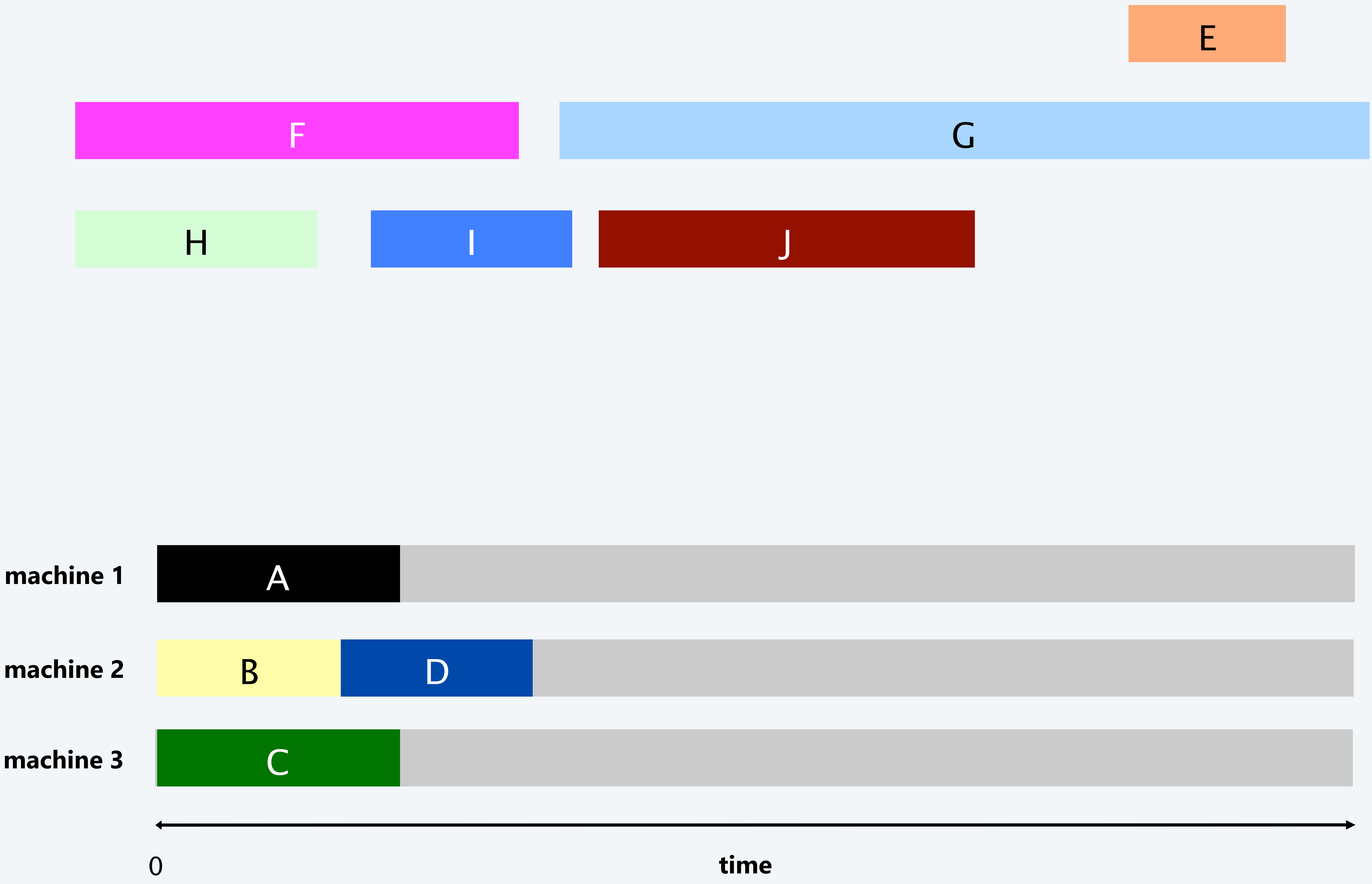
List scheduling demo



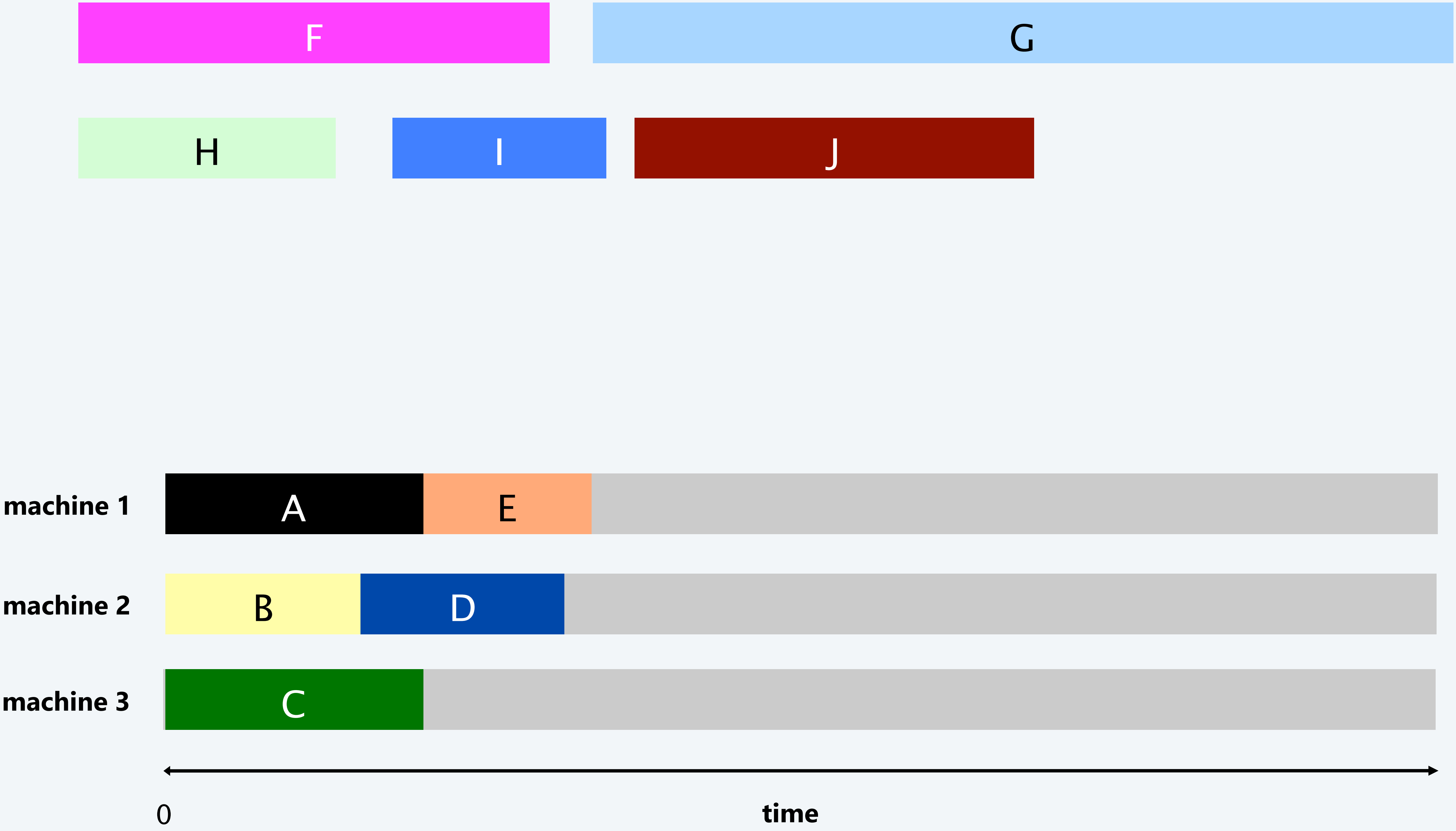
List scheduling demo



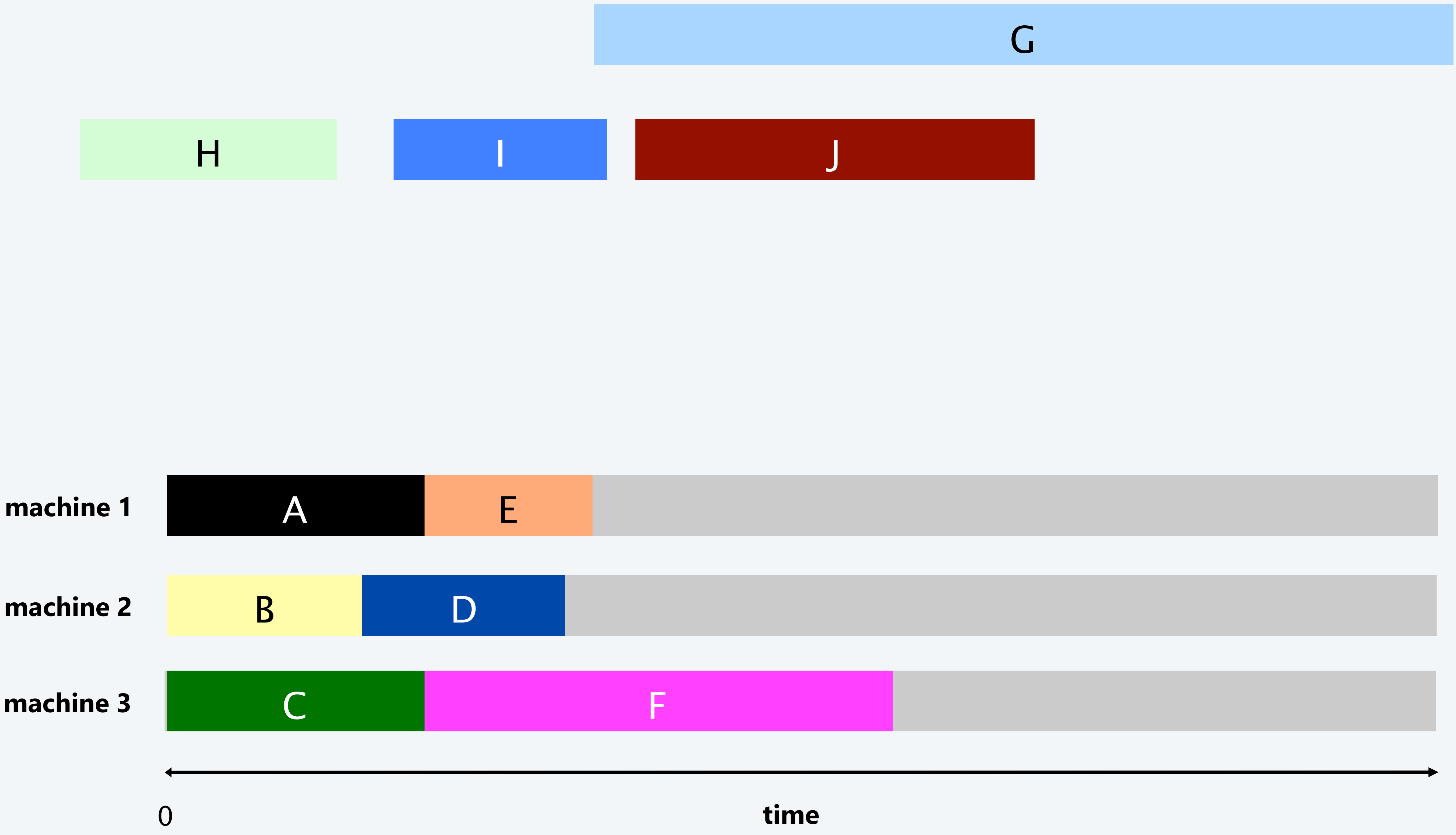
List scheduling demo



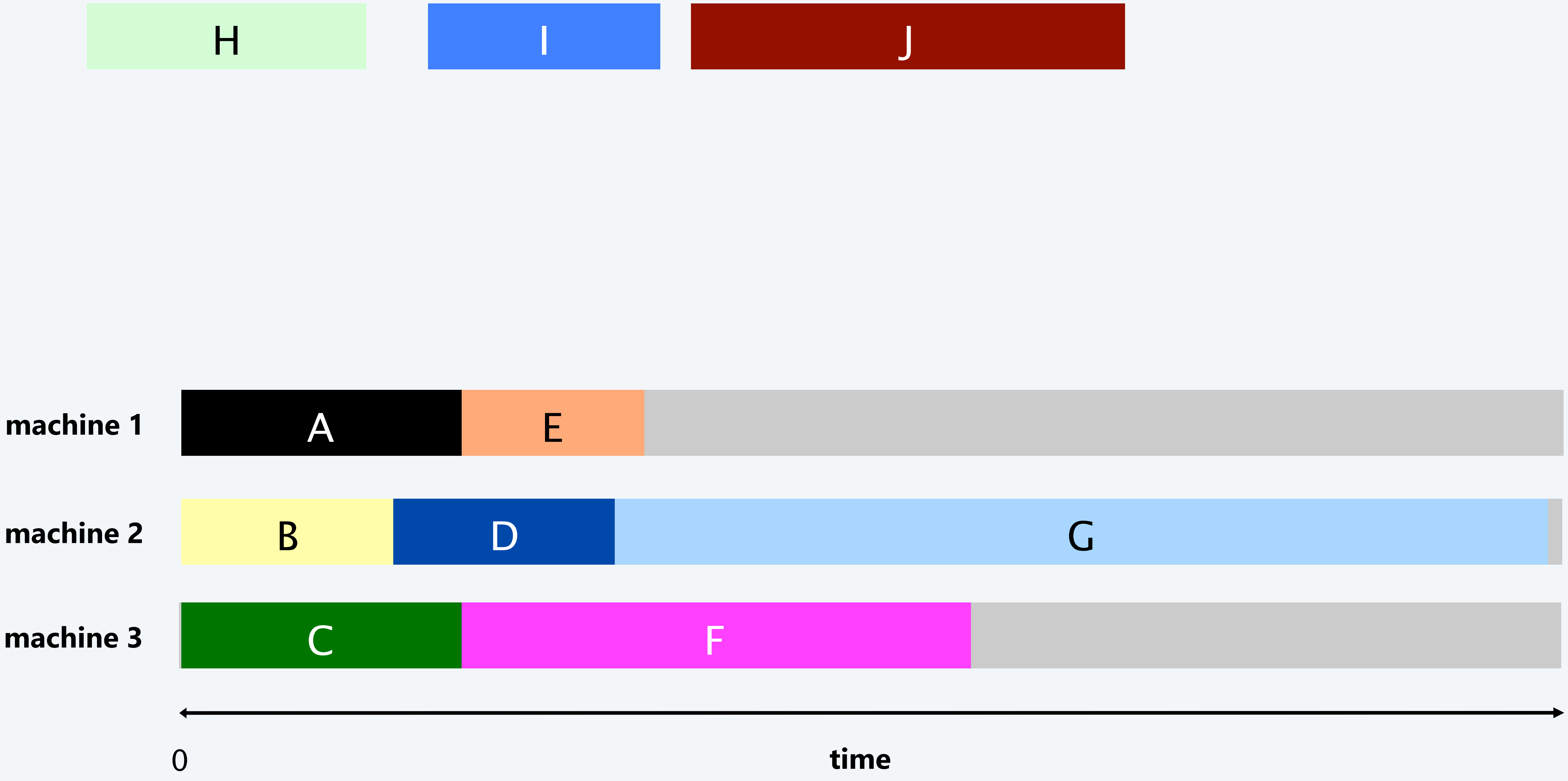
List scheduling demo



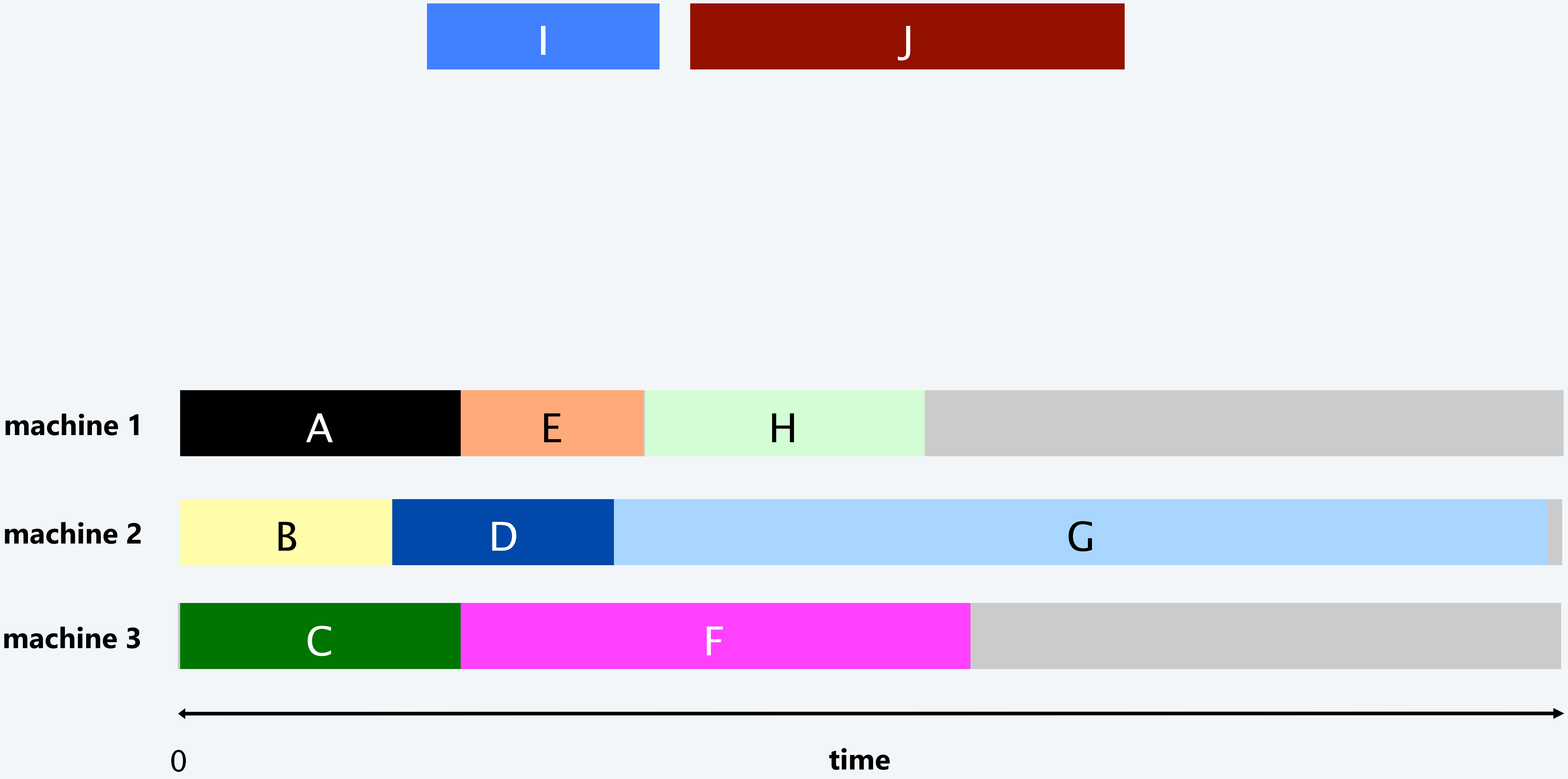
List scheduling demo



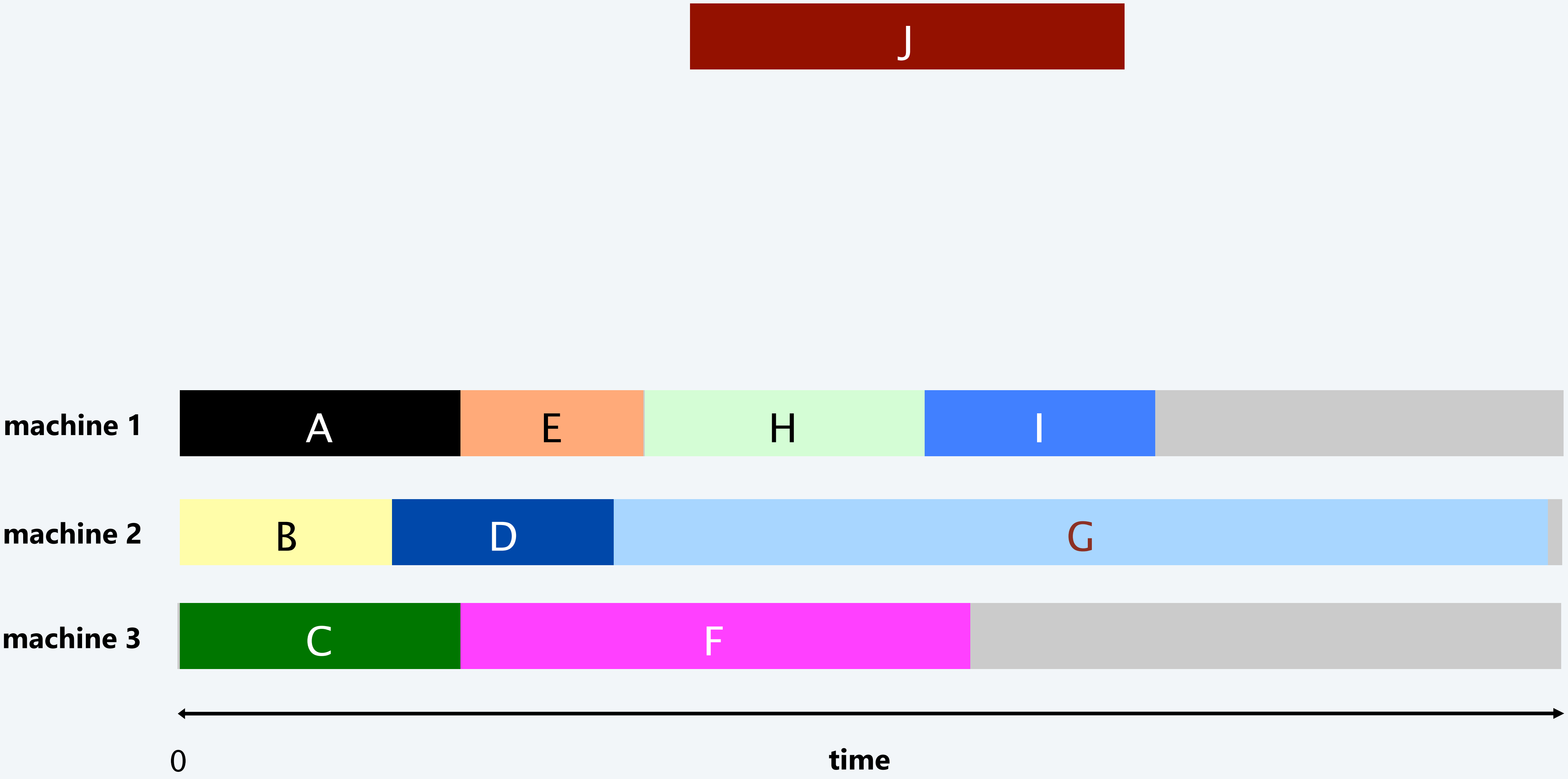
List scheduling demo



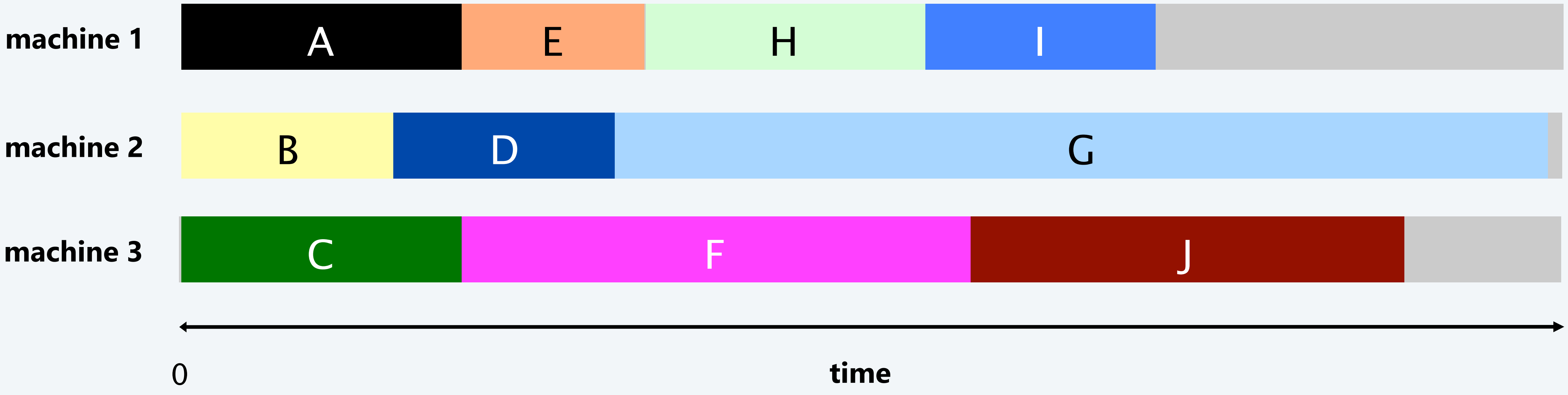
List scheduling demo



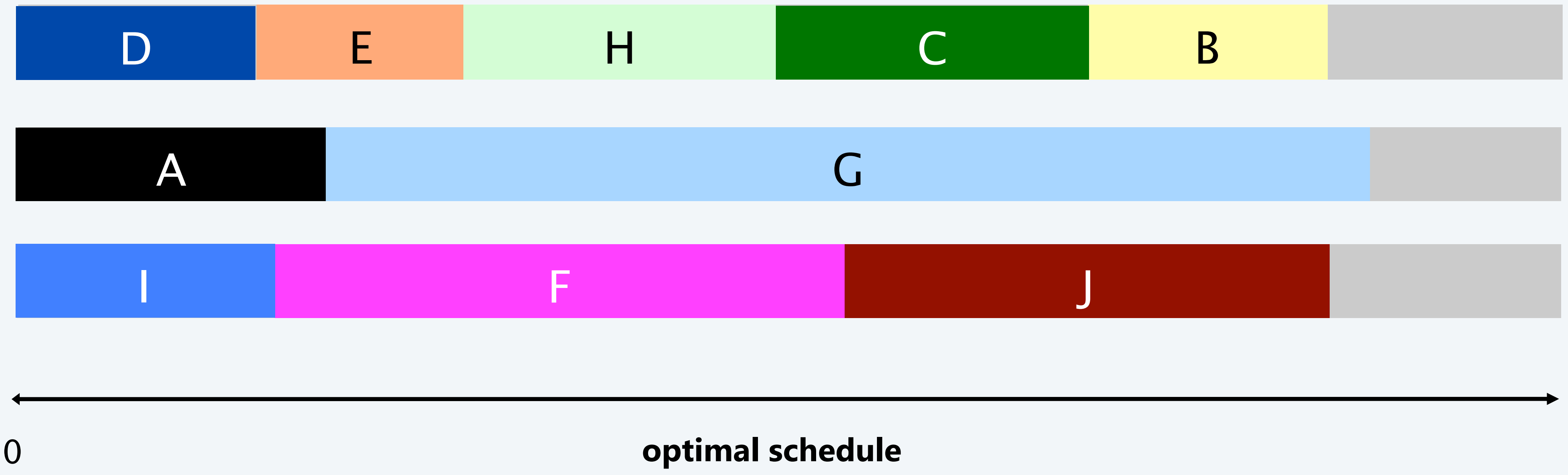
List scheduling demo

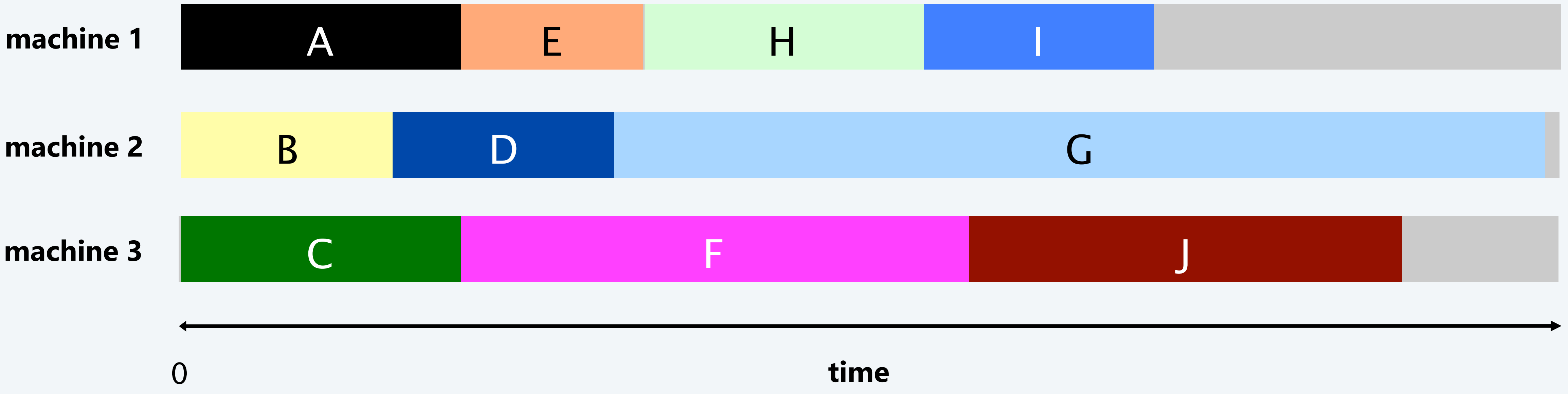


List scheduling demo



List scheduling demo





Load balancing: list scheduling analysis

Theorem. [Graham 1966] Greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L^*.

Lemma 1. For all k : the optimal makespan $L^* \geq t_k$.

Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. The optimal makespan $L^* \geq \frac{1}{m} \sum_k t_k$.

Pf.

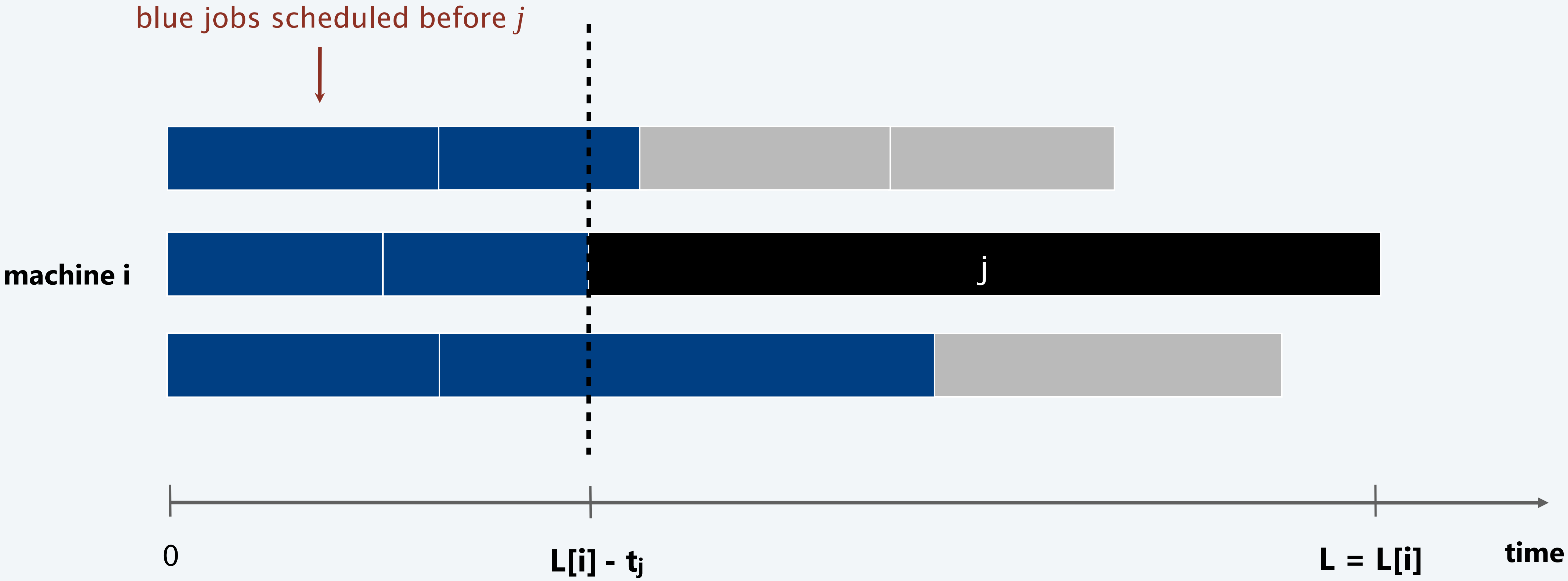
- The total processing time is $\Sigma_k \, t_k$.
- One of m machines must do at least a $1 / m$ fraction of total work. ▪

Load balancing: list scheduling analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L[i]$ of bottleneck machine i. ← machine that ends up with highest load

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load.
  Its load before assignment is $L[i] - t_j$; hence $L[i] - t_j \leq L[k]$ for all $1 \leq k \leq m$.



Load balancing: list scheduling analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L[i]$ of bottleneck machine i. ← machine that ends up with highest load

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L[i] - t_j$; hence $L[i] - t_j \le L[k]$ for all $1 \le k \le m$.
- Sum inequalities over all k and divide by m:

$$\begin{aligned} L[i] - t_j &\le \frac{1}{m} \sum_k L[k] \\ &= \frac{1}{m} \sum_k t_k \\ \text{Lemma 2} \longrightarrow \quad &\le L^*. \end{aligned}$$

- Now, $L = L[i] = \underbrace{(L[i] - t_j)}_{\le L^*} + \underbrace{t_j}_{\le L^*} \le 2L^*$.

above inequality

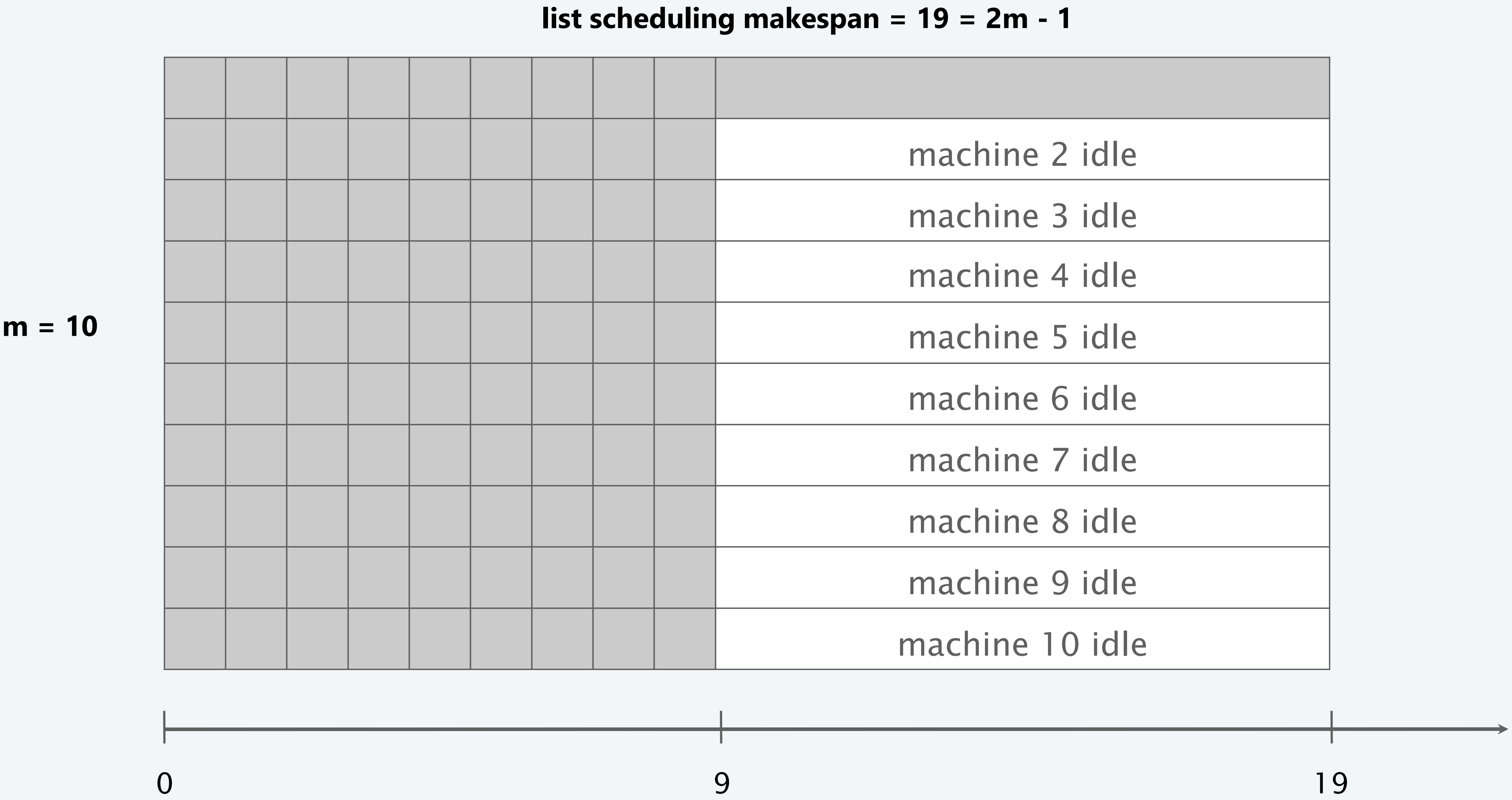
Lemma 1

Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, first $m\,(m-1)$ jobs have length 1, last job has length m.

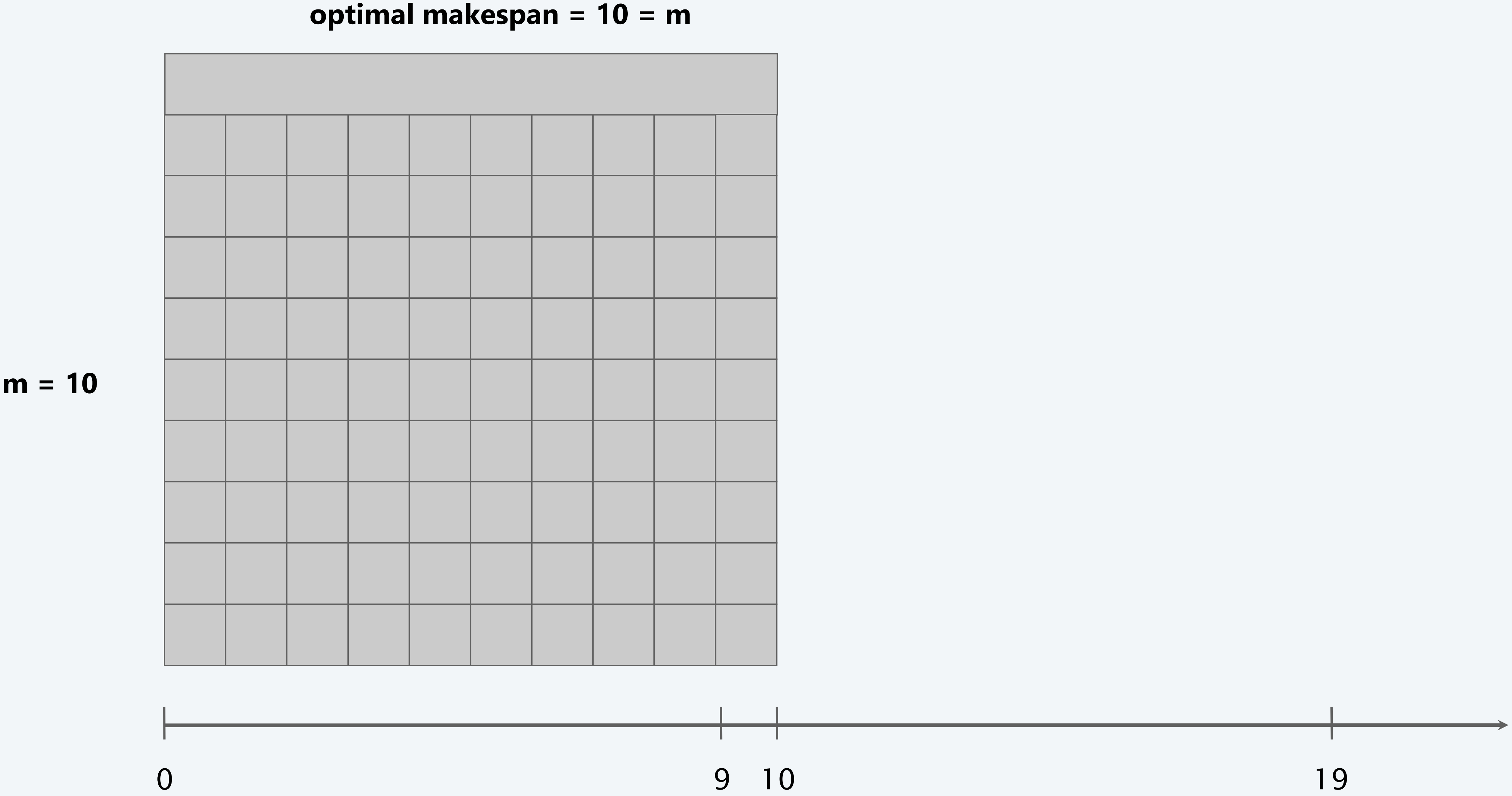


Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, first $m\ (m-1)$ jobs have length 1, last job has length m.



Load balancing: LPT rule

Longest processing time (LPT). Sort n jobs in decreasing order of processing times; then run list scheduling algorithm.

```
LPT-LIST-SCHEDULING $(m, n, t_1, t_2, \ldots, t_n)$

SORT jobs and renumber so that $t_1 \geq t_2 \geq \ldots \geq t_n$.

FOR $i = 1$ TO $m$
    $L[i] \leftarrow 0.$                      ⟵ load on machine i
    $S[i] \leftarrow \varnothing.$            ⟵ jobs assigned to machine i

FOR $j = 1$ TO $n$
    $i \leftarrow \text{argmin}_k\, L[k].$    ⟵ machine i has smallest load
    $S[i] \leftarrow S[i] \cup \{ j \}.$      ⟵ assign job j to machine i
    $L[i] \leftarrow L[i] + t_j.$             ⟵ update load of machine i

RETURN $S[1], S[2], \ldots, S[m].$
```

Load balancing: LPT rule

Observation. If bottleneck machine i has only 1 job, then optimal.

Pf. Any solution must schedule that job. ▪

Lemma 3. If there are more than m jobs, $L^* \geq 2\, t_{m+1}$.

Pf.

- Consider processing times of first $m+1$ jobs $t_1 \geq t_2 \geq \ldots \geq t_{m+1}$.
- Each takes at least t_{m+1} time.
- There are $m+1$ jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. ▪

Theorem. LPT rule is a 3/2-approximation algorithm.

Pf. [similar to proof for list scheduling]

- Consider load $L[i]$ of bottleneck machine i.
- Let j be last job scheduled on machine i. ← assuming machine i has at least 2 jobs, we have $j \geq m+1$

$$L = L[i] = \underbrace{(L[i] - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2} L^*} \leq \frac{3}{2} L^* \text{ ▪}$$

as before → $\leq L^*$

$\leq \frac{1}{2} L^*$ ← Lemma 3 (since $t_{m+1} \geq t_j$)

Load balancing: LPT rule

Q. Is our 3/2 analysis tight?

A. No.

Theorem. [Graham 1969] LPT rule is a 4/3-approximation.

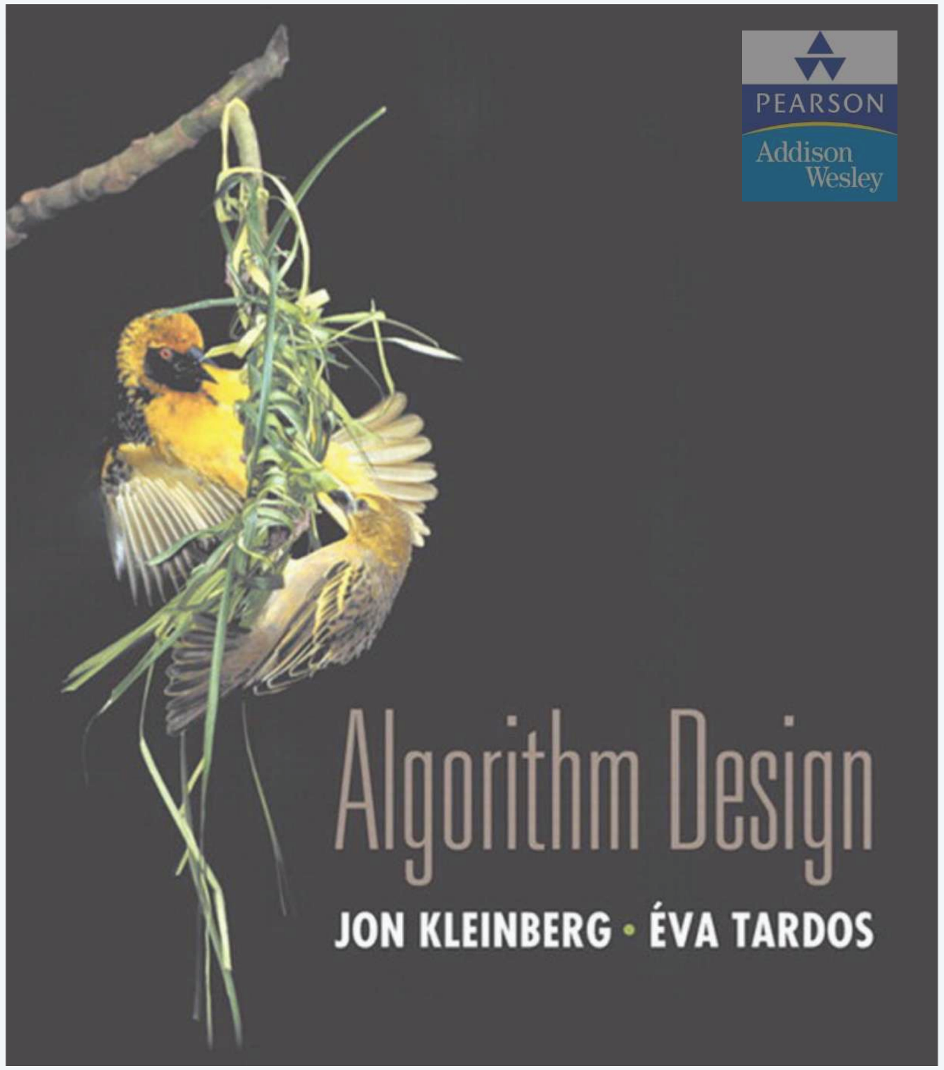
Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis tight?

A. Essentially yes.

Ex.

- m machines
- $n = 2m + 1$ jobs
- 2 jobs of length m, $m + 1$, …, $2m - 1$ and one more job of length m.
- Then, $L / L^* = (4m - 1) / (3m)$



SECTION 11.2

11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*