# CS528
# Task Scheduling

A Sahu
Dept of CSE, IIT Guwahati

# Outline

- $P_m \mid p_j, pmtn \mid C_{max}$ : Linear time solution
- $Q_m \mid p_j, pmtn \mid C_{max}$ : Poly time solution
- $Q_m \mid ptmn \mid \Sigma C_j$ Optimal Solution
- $P_m \mid p_j \mid C_{max}$
  - **ILP Solution : Exponential**
  - 2 Approx, 2-1/m approx.
  - LPT : 3/2 and 4/3 Approx
- $P_m \mid p_j=1 \mid \Sigma w_j U_j$ Optimal Solution
- $P_m \mid p_j \mid \Sigma U_j$ NPC, Heuristic and Counter example
- $P_m \mid pmtn, p_j \mid \Sigma U_j$ **in NPC**
- $P_m \mid prec, p_j = 1 \mid C_{max}$ **in NPC**
  - 2 Approx

# Detail of
# Minimum Makespan Scheduling

# $Pm|\ p_j|\ Cmax$

# $Pm|\ p_j|\ Cmax$
# Minimum makespan scheduling

- $P_m|p_j|C_{max}$ in NPC
- Given processing times for n jobs, $p_1, p_2, \ldots, p_n$, and an integer m
- Find an assignment of the jobs to m identical machines
- So that the completion time, also called the makespan, is minimized.

# 0-1 Linear Programming Solution to Scheduling Problem

$x_{ij} = \{0, 1\}$ whether job j is scheduled in machine i

$$\min \quad T$$

$$\sum_{i=1}^{m} x_{ij} = 1$$ for each job j

Each job is scheduled in one machine.

$$\sum_{j=1}^{n} x_{ij} \cdot p_{ij} \leq T$$ for each machine i

Each machine can finish its jobs by time T

$$0 \leq x_{ij}$$ for each job j, machine i

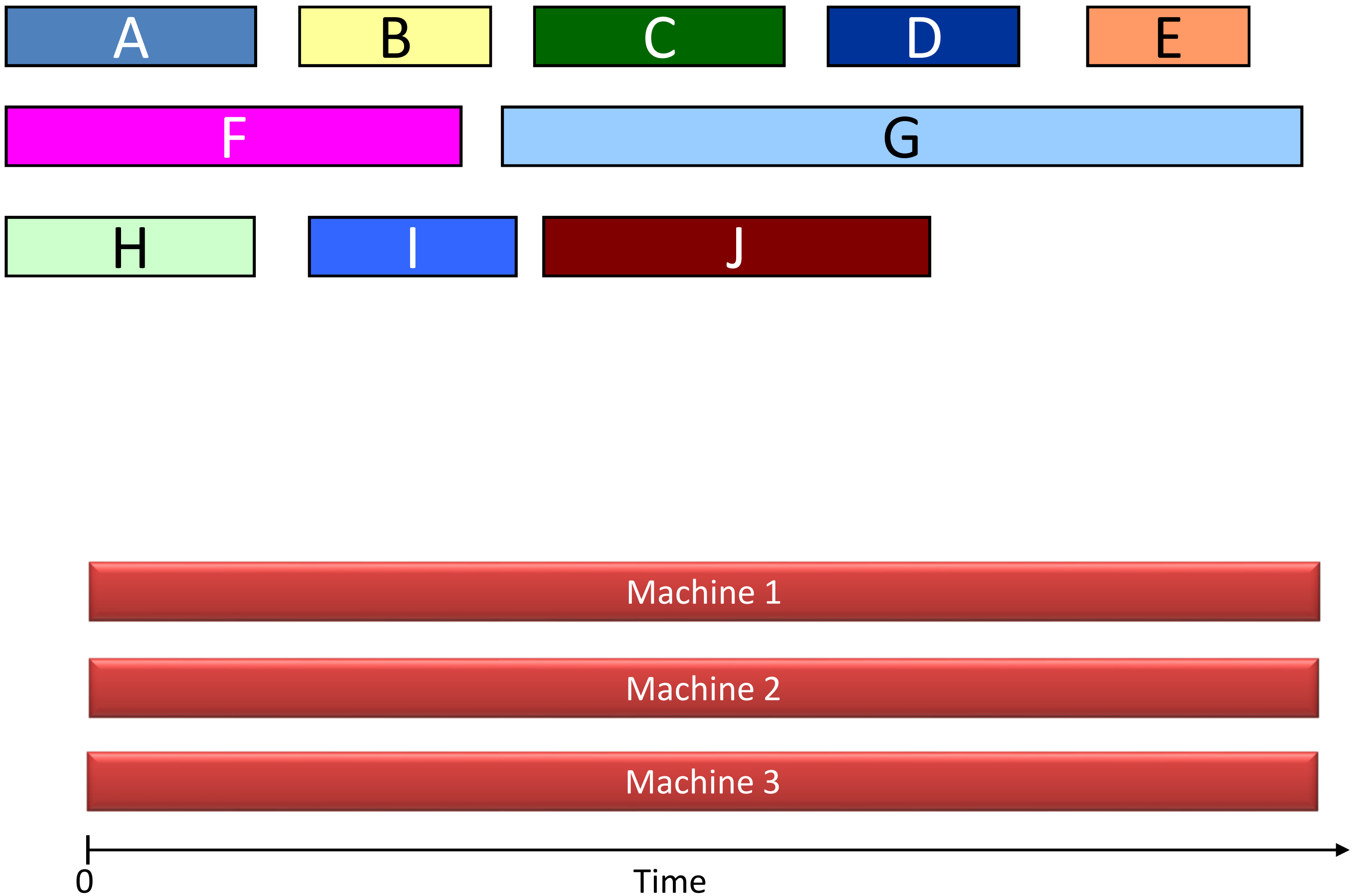
# Minimum makespan scheduling: Arbitrary List

- List Scheduling : Approximation
- Algorithm
  - 1. Order the jobs arbitrarily.
  - 2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.
- Above algorithm achieves an approximation guarantee of 2

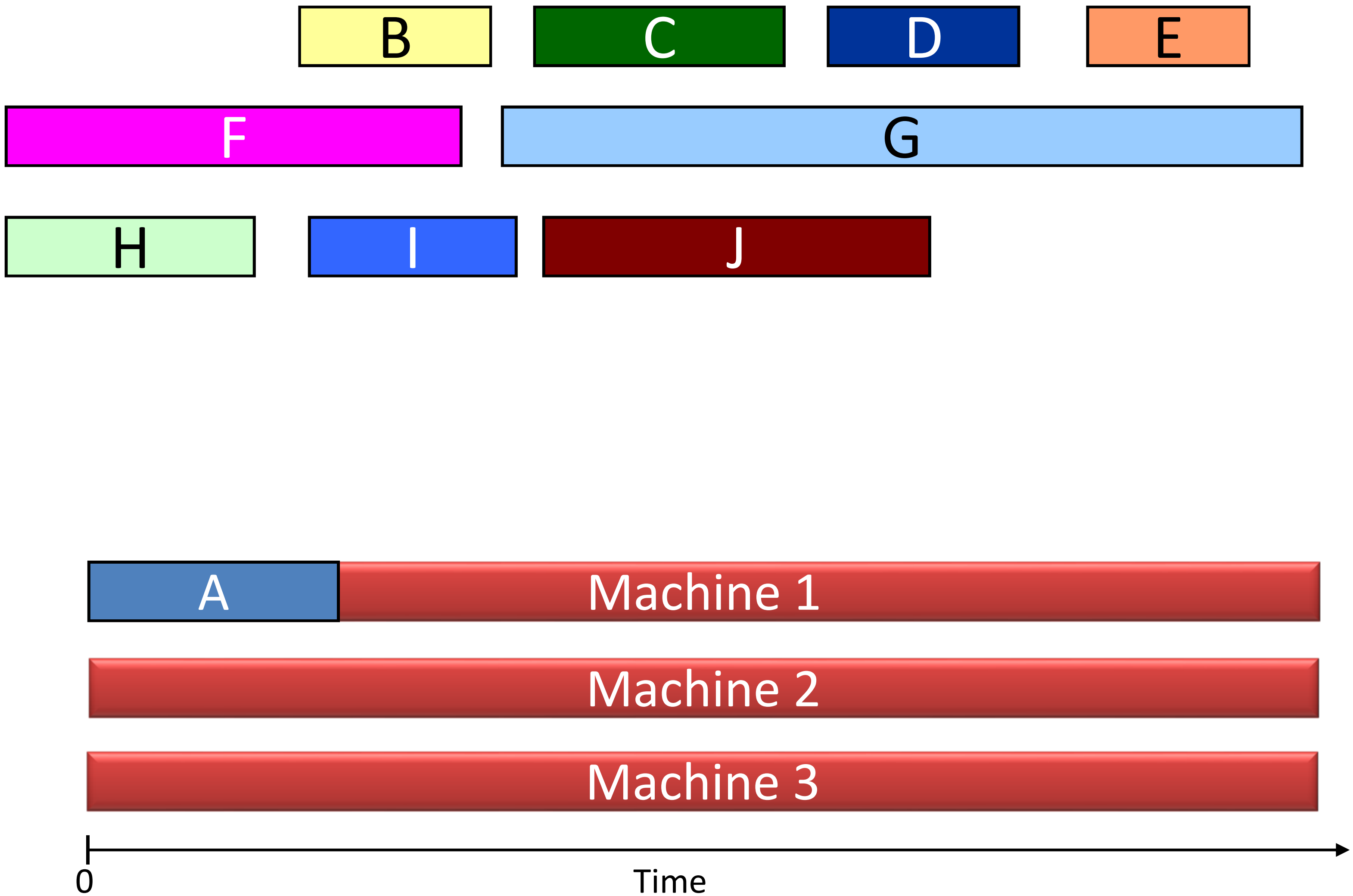
# Minimum Makespan scheduling: Arbitrary List

- List Scheduling : Approximation
- Algorithm
  - 1. Order the jobs arbitrarily.
  - 2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.
- Above algorithm achieves an approximation guarantee of 2

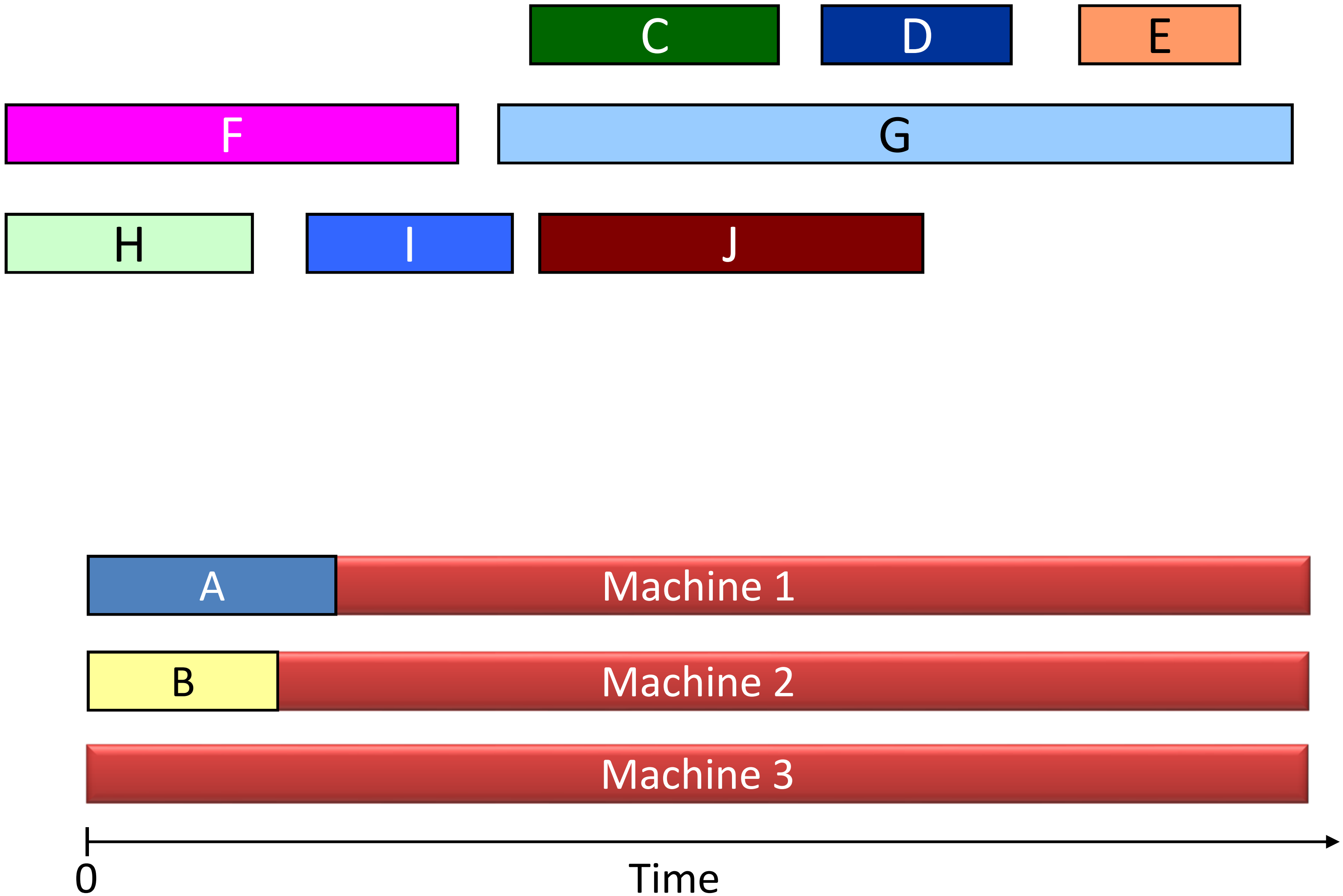
# Load Balancing: List Scheduling



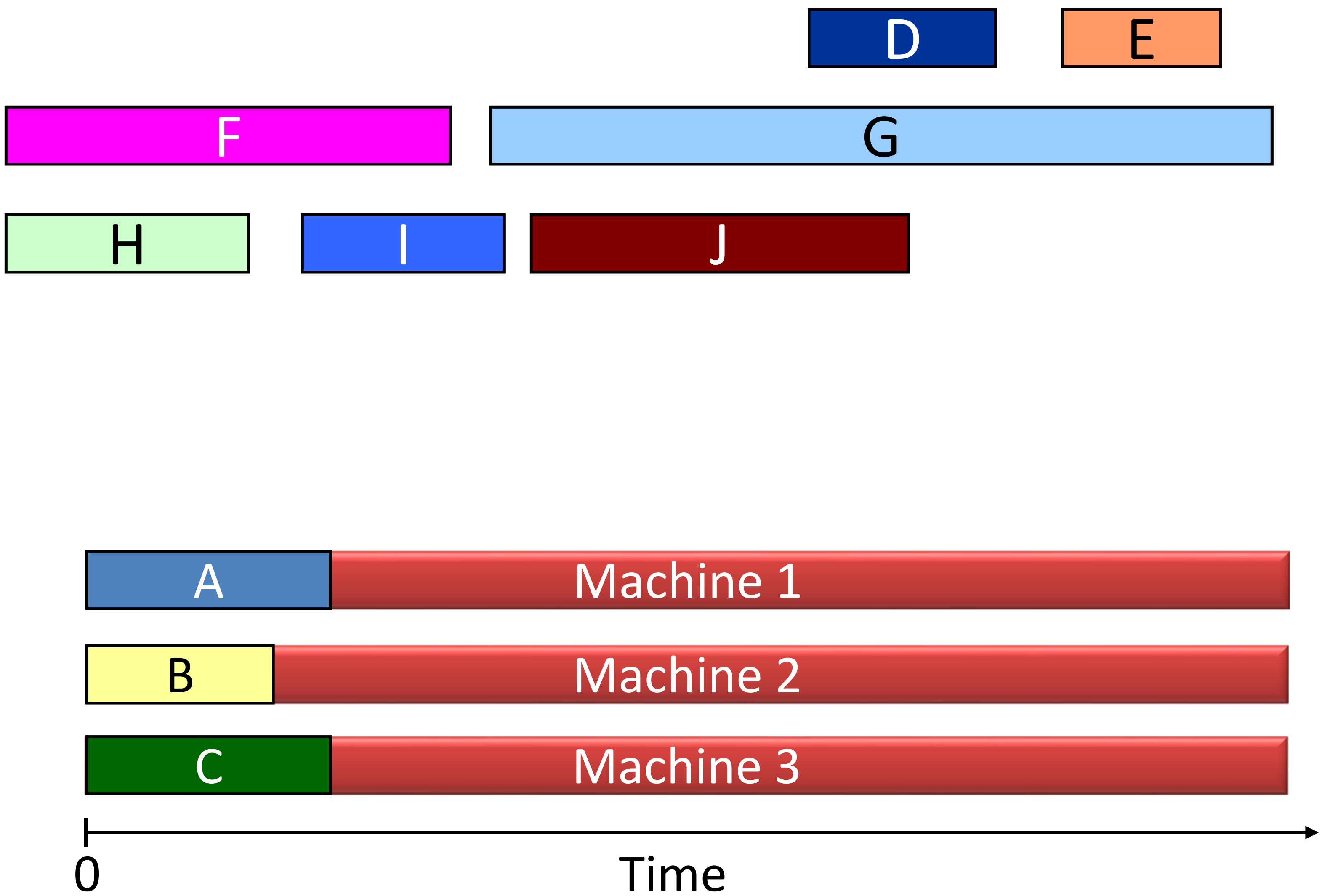
# Load Balancing: List Scheduling



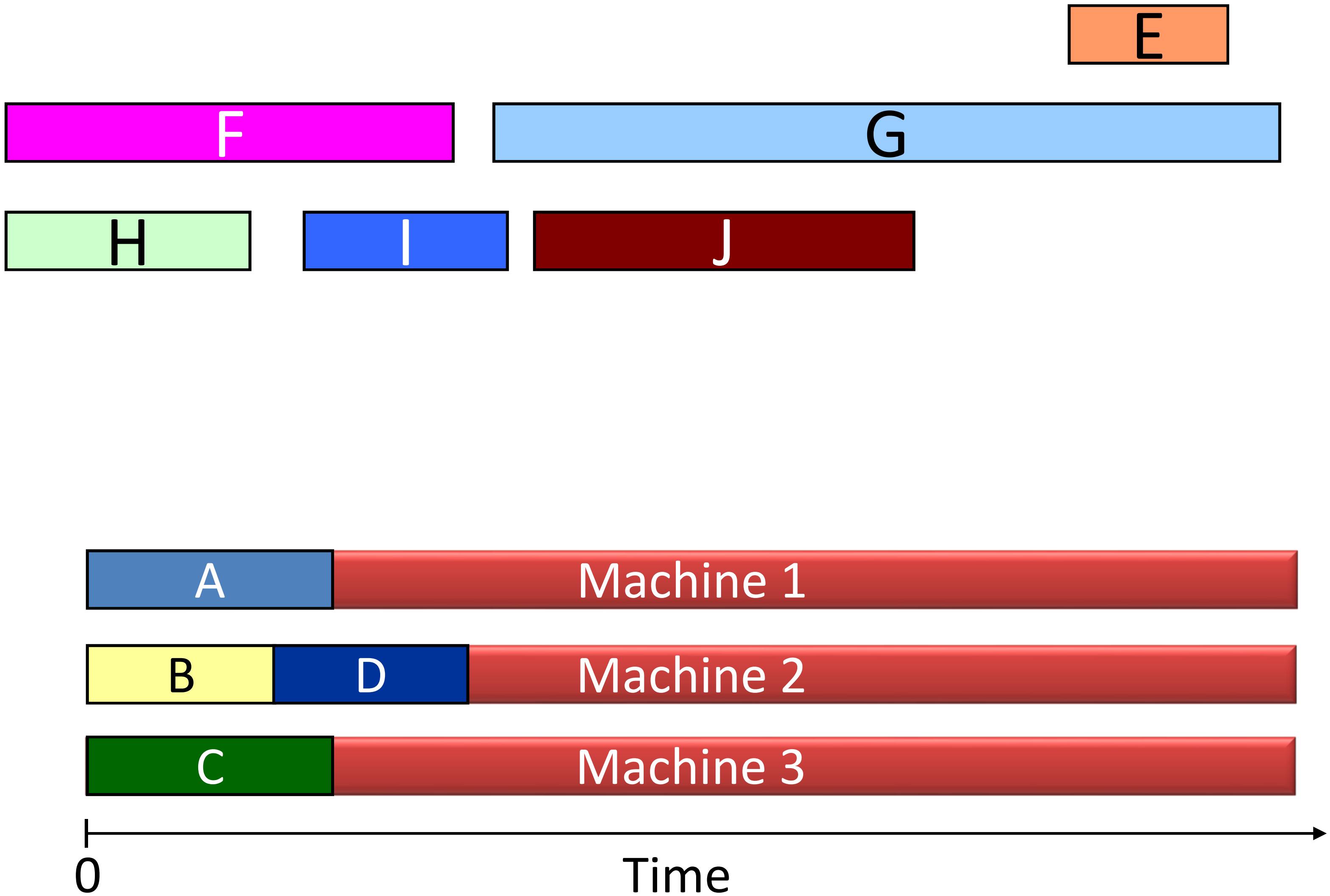
# Load Balancing: List Scheduling



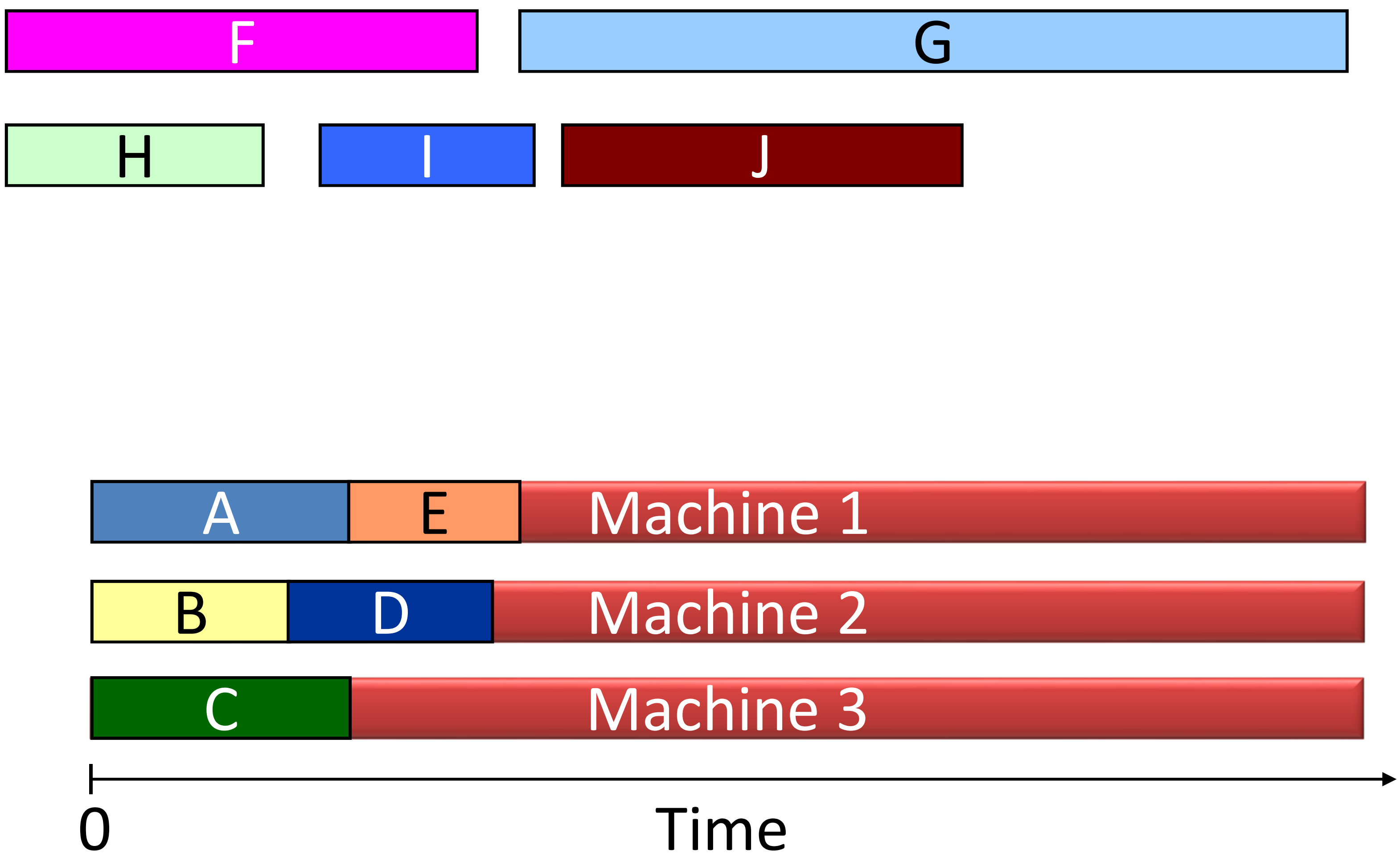
# Load Balancing: List Scheduling



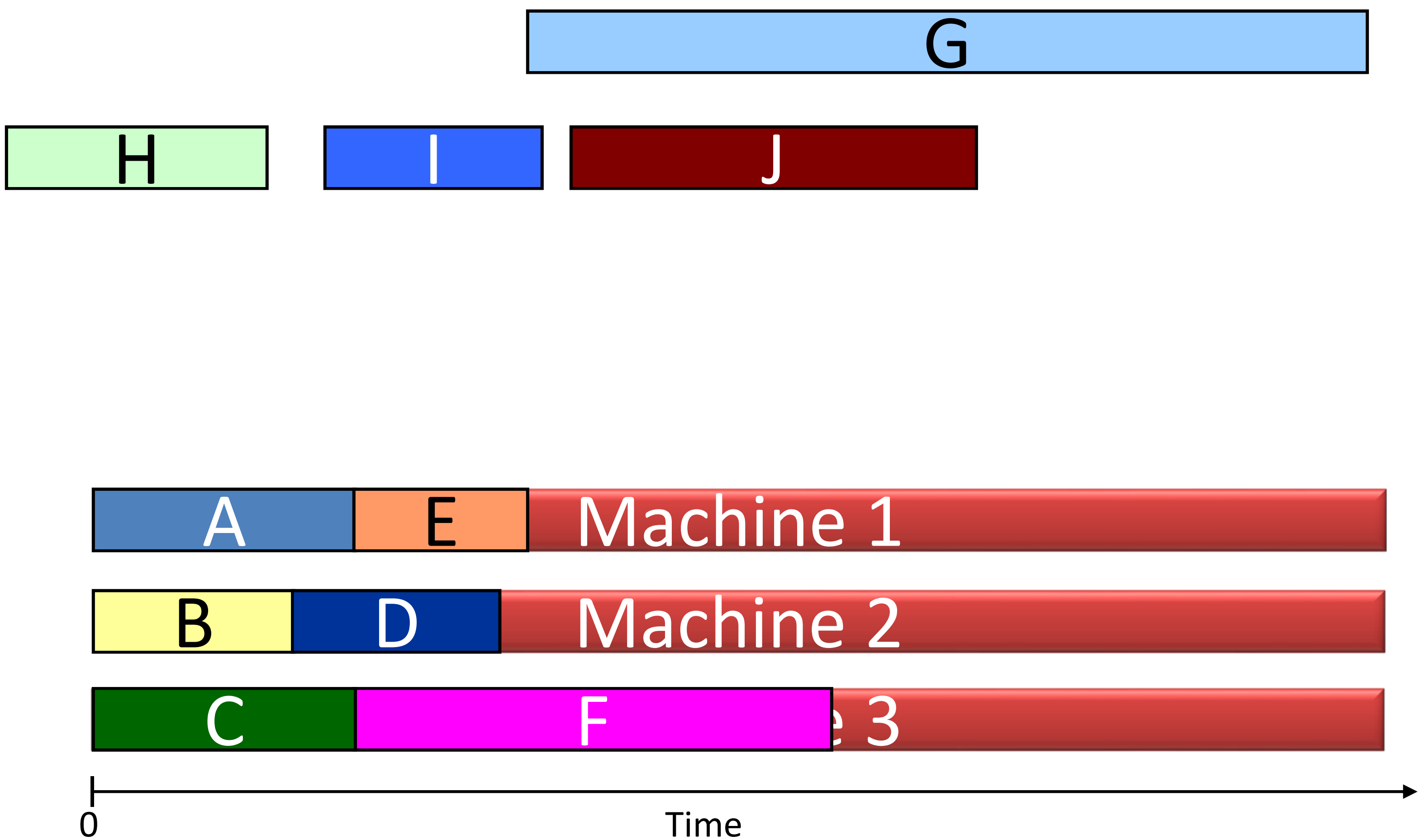
# Load Balancing: List Scheduling



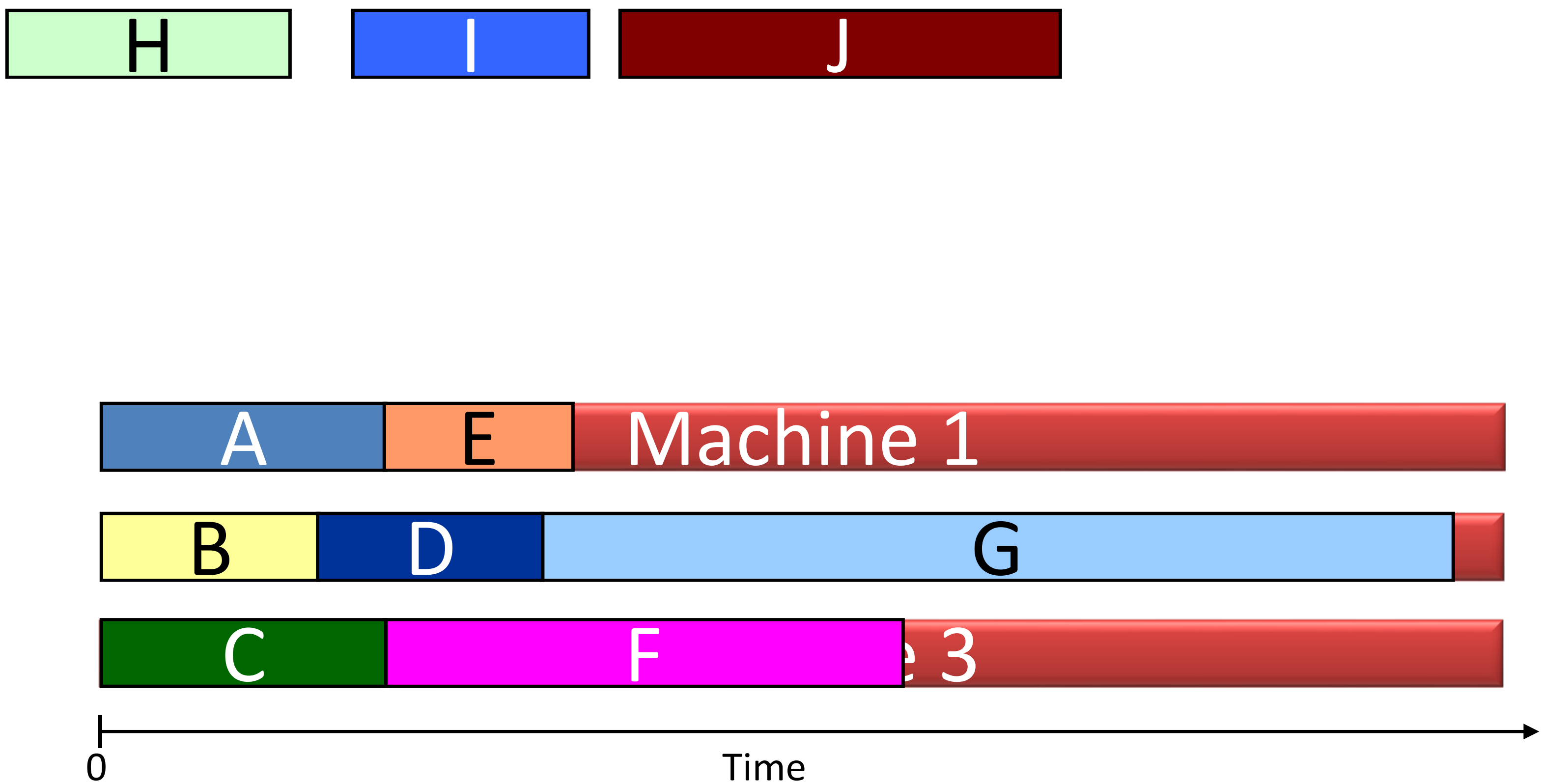
# Load Balancing: List Scheduling



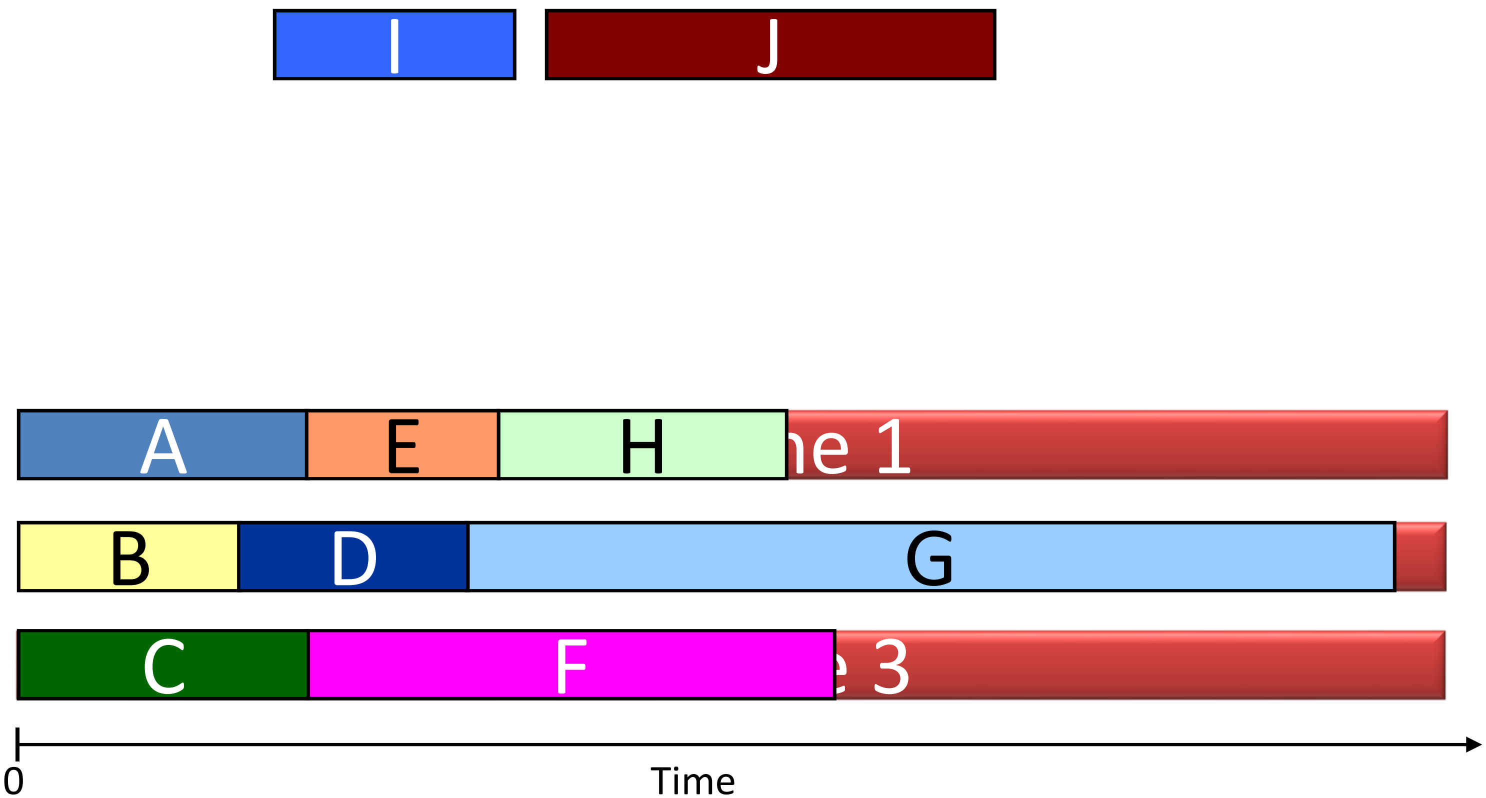
# Load Balancing: List Scheduling



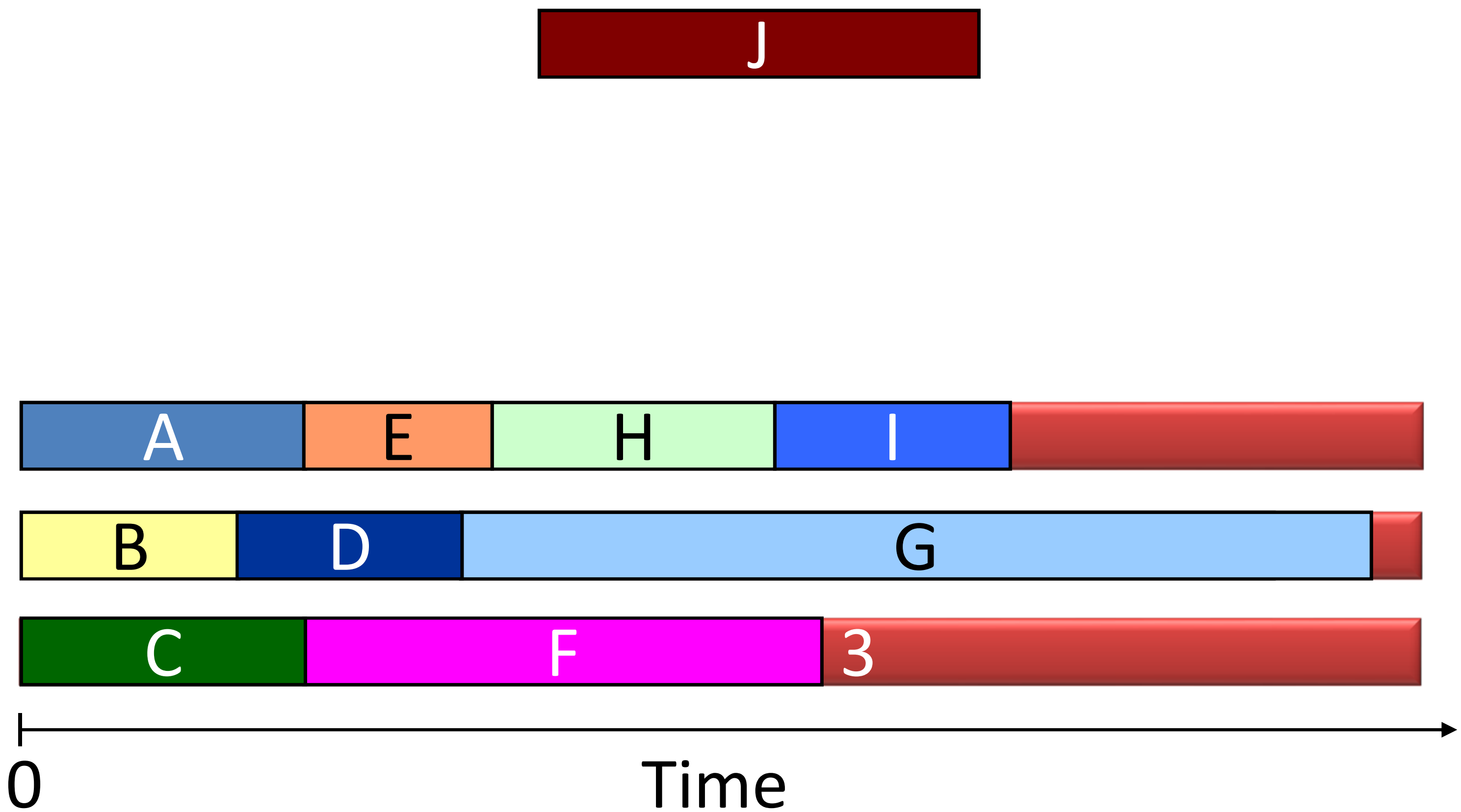
# Load Balancing: List Scheduling



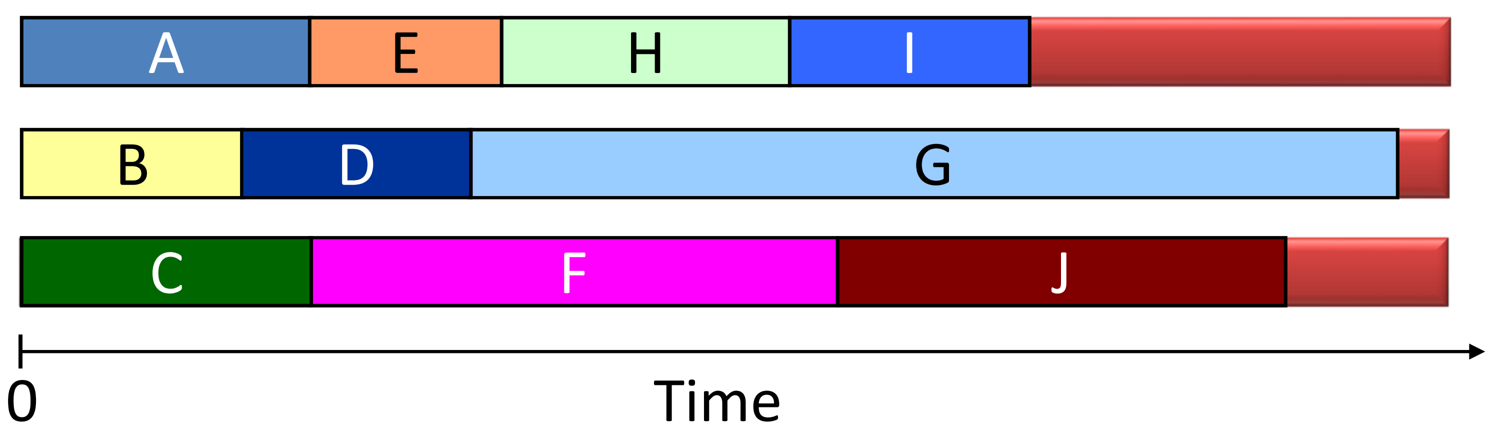
# Load Balancing: List Scheduling



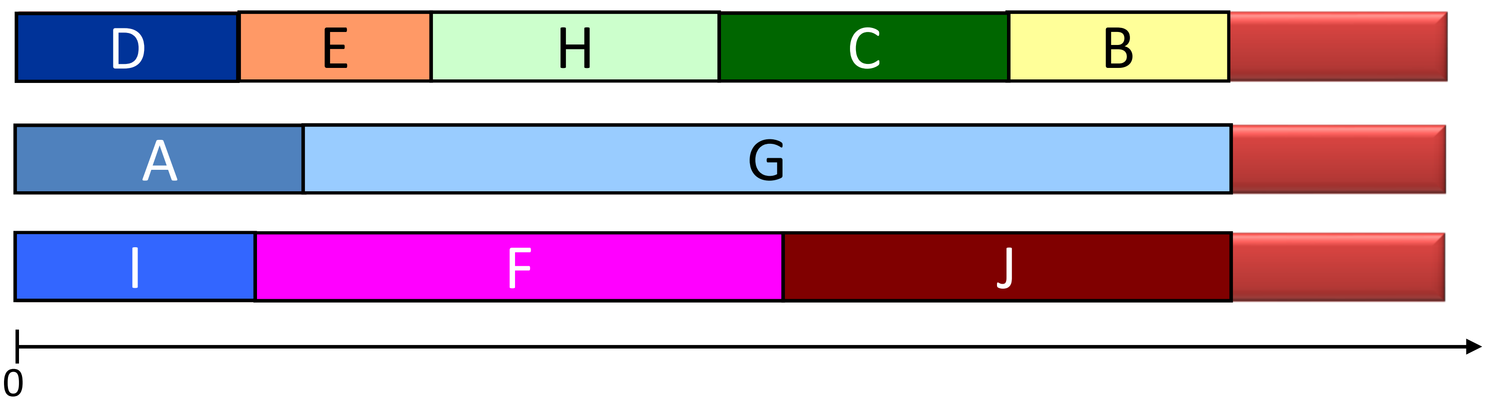
# Load Balancing: List Scheduling



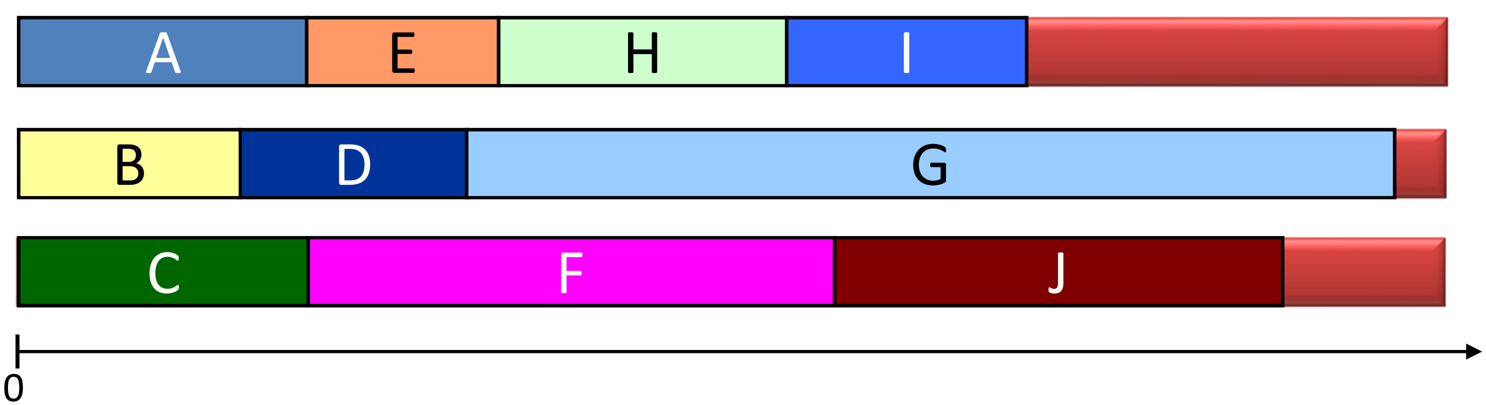
# Load Balancing: List Scheduling



# Load Balancing:  List Scheduling




Optimal Schedule




List schedule

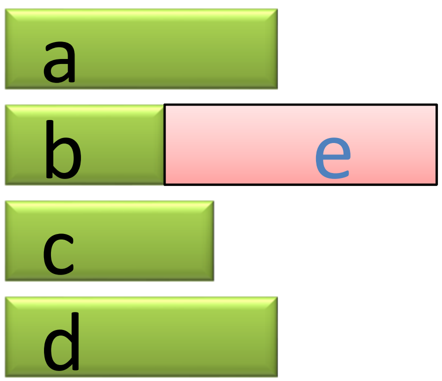
# LS is 2 APPRX

# LS is 2 APPRX

**Algorithm: List scheduling**

**Basic idea: In a list of jobs,**

**schedule the next one as soon as a machine is free**




machine 1

machine 2

machine 3

machine 4

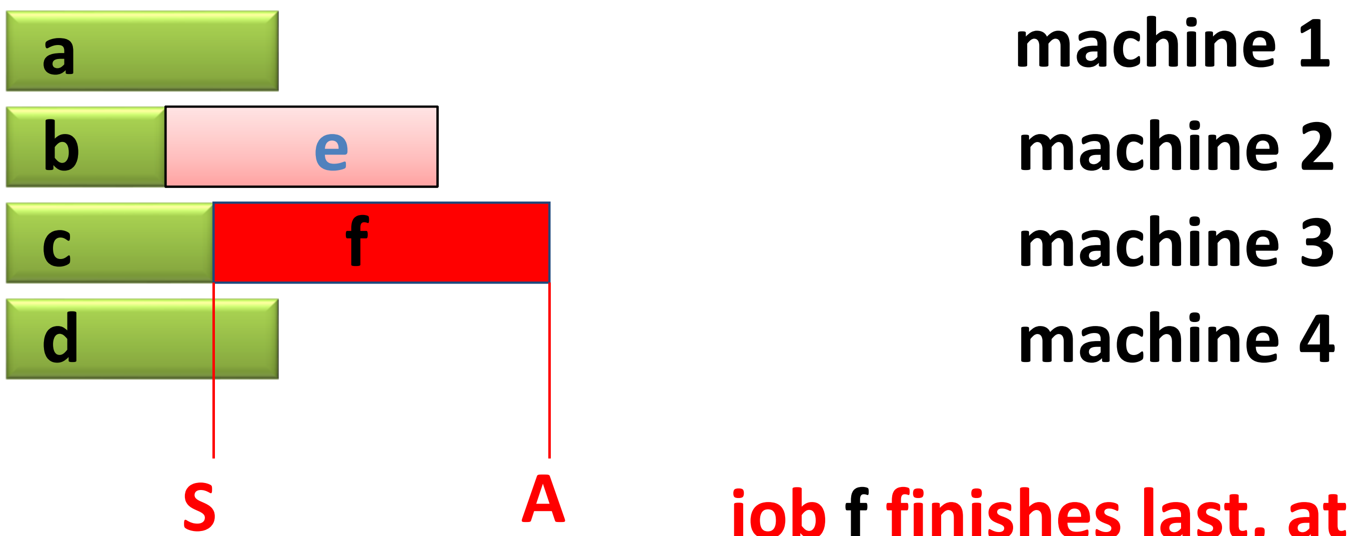
Good or bad ?

# List Scheduling is "2-approximation" (Graham, 1966)

**Algorithm:** **List scheduling**

**Basic idea: In a list of jobs,**

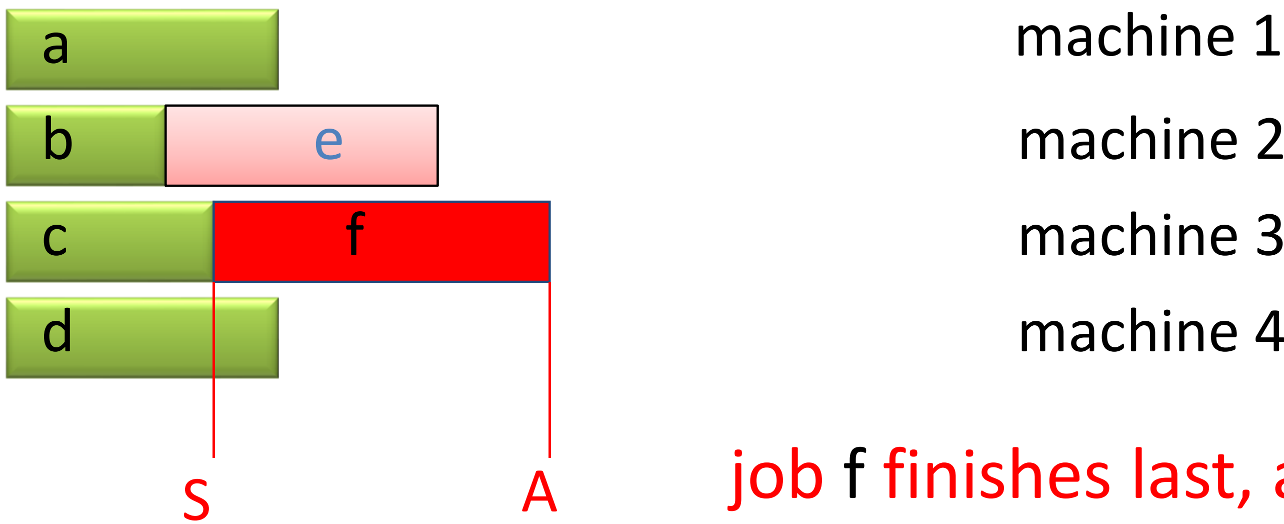
**schedule the next one as soon as a machine is free**




**job f finishes last, at time A**

**compare to time OPT of best schedule: how ?**

# List Scheduling is "2-approximation"




job f finishes last, at time A

compare to time OPT of best schedule: how ?

(1) job f must be scheduled in the best schedule at some time:
**f <= OPT.** ➔ **A – S <= OPT.**

(2) up to time S, all machines were busy all the time, and OPT cannot beat that, and job f was not yet included: **S < OPT.**

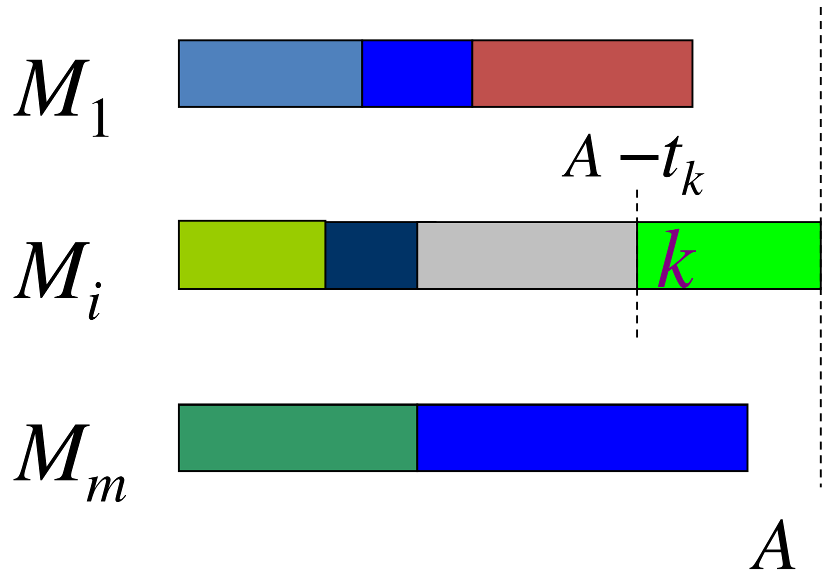
(3) both together: **A = A – S + S = (A-S) +S < 2*OPT.**

"2-approximation" (Graham, 1966)

# LS is (2-1/m) APPRX

# LS achieves a perf. ratio 2–1/*m*.

So all machines are busy from time 0 through $A$-$t_k$
Consequently,




Let $\boldsymbol{T} = \sum \boldsymbol{t_i}$, $i$=1,2…,$n$

$$T\text{-}t_k \geq m(A - t_k) \quad \rightarrow \; T\text{-}t_k \geq mA - mt_k$$

$$\rightarrow \quad T\text{-}t_k + mt_k \geq mA \;\rightarrow\; T + (m\text{-}1)t_k \geq mA$$

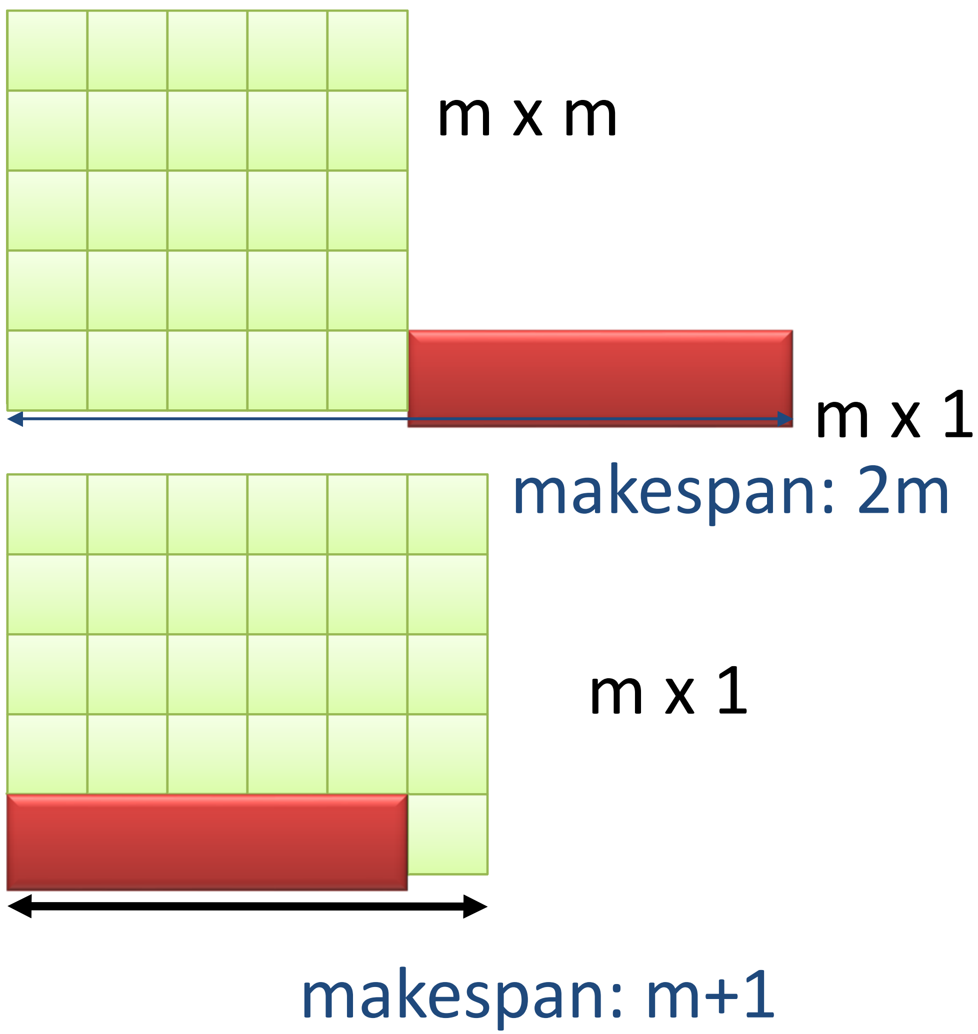
*As* *m*. T* ≥ T. So, T*≥$T/m$.
Also T*≥$t_k$ for every $k$.

So, $$A \leq T/m + t_k\,(m\text{-}1)/m$$

$$\leq \; T^* + (1\text{-}1/m)\, T^*$$

$$\boldsymbol{A \leq (2\text{-}1/m)\, T^*}$$

# Example: Worst Case



# LPT Rule: List with LPT

- List scheduling can do badly if long jobs at the end of the list spoil an even division of processing times.
- We now assume that the jobs are all given ahead of time, i.e. the LPT rule works only in the off-line situation. Consider the "***Largest Processing Time first***" or LPT rule that works as follows.

# LPT Rule: List with LPT

LPT Algorithm

1 sort the jobs in order of decreasing processing times: $t_1 \geq t_2 \geq . . . \geq t_n$

2 execute list scheduling on the sorted list

3 **return** the schedule so obtained.

- The LPT rule achieves 3/2-Approx **Sec 11.1 of Eva Tardos Algo Book, Appx Algo Chapter**

- The LPT rule achieves a performance ratio $4/3-1/(3m)$. **Prove out of Syllabus**

# LPT 3/2-Approx: Jobs are sorted

- Job Time: $t_1 \geq t_2 \geq t_3 \geq \ldots \geq t_j$
- Suppose j (=m+1) jobs ( j>m), in LPT $T^* \geq 2.t_{m+1}$

- Examples: m =5 , j =6

 10, 9, 8,7, 5, 4,...

 $t_{m+1}=4$

 $T^* \geq 2*4=8$

# LPT 3/2-Approx: Jobs are sorted

- Job Time: $t_1 \geq t_2 \geq t_3 \geq \ldots \geq t_j$
- Suppose j (=m+1) jobs ( j>m), in LPT $T^* \geq 2.t_{m+1}$
- Suppose a machine $M_i$ have at least two jobs and $t_j$ be last job ($j \geq m+1$) assigned to $M_i$

$$t_j \leq t_{m+1} \leq T^*/2$$

- Also we have $t_j \leq T^*$ and $T_i - t_j \leq T^*$, where $T_i$ is sum of ET of task assigned to $M_i$
- $T_i - t_j \leq T^* \rightarrow T_i \leq T^* + t_j \rightarrow T_i \leq T^* + T^*/2$

$$T_i \leq (3/2)\, T^*$$

# Scheduling
# of
# Independent Tasks with Deadline

# $P|p_j=1|\Sigma w_j U_j$

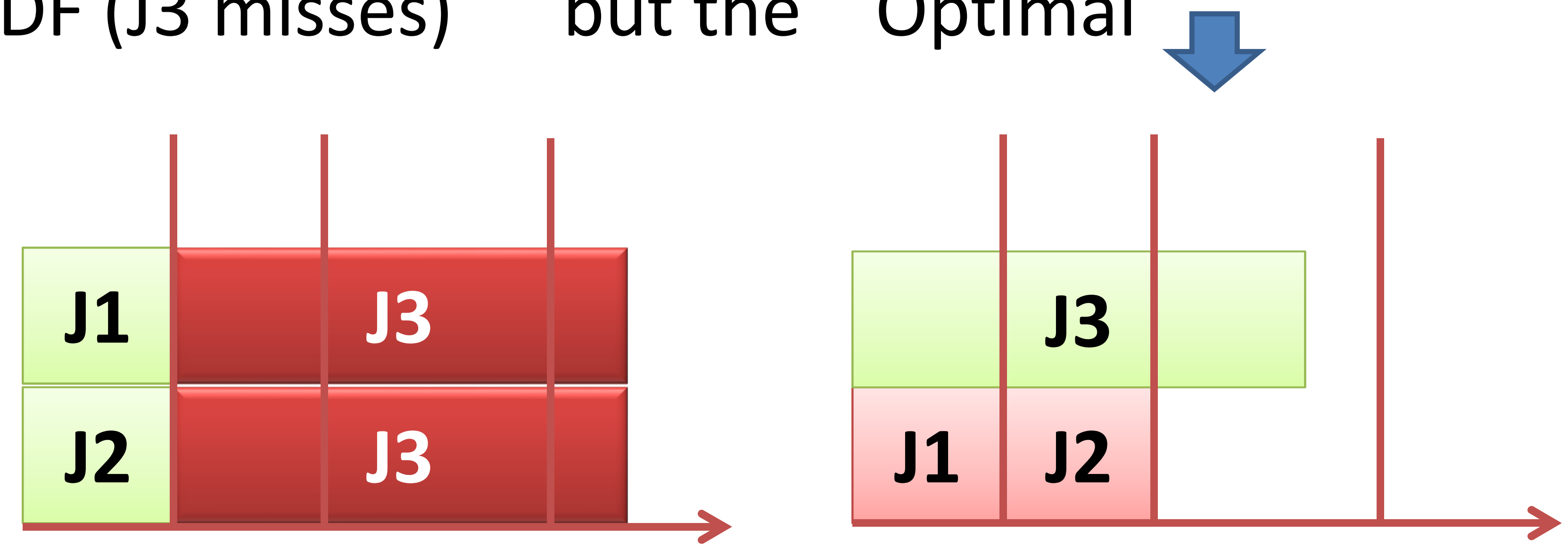
- Sorting task based on $d_i$ and $d_1 \leq d_2 \leq \ldots \leq d_n$
- Approach 1: Simply scheduling and rejecting the unfit task will not minimize $w_i$
  - **Will not work : you need to take care of weight**
- Approach 2: Sorting task based on $w_i/d_i$
  - Gives priority of task with higher weight but
  - Simply may reject a task based on deadline
  - **Will not work : for optimality**

# $P|p_j=1|\Sigma w_j U_j$

- Sort all the jobs with $d_1 \leq d_2 \leq \ldots \leq d_n$
- Set $S=\Phi$
- For i=1 to n do
  - If ($i_{th}$ task is late when scheduled in the earliest time slot on a machine)
    - Find a task i* with $w_i$* = min weight of tasks in the already scheduled tasks of the set S
    - If ($w_i$* < $w_i$) replace i* with $i_{th}$ task in the schedule and in S.
  - else add $i_{th}$ task to S and schedule the task in the earliest time slot

# $P||\Sigma U_j$

- NPC: Sorting based on deadlines is excellent heuristics for most of the case, Experimentally
- But not optimal
- Counter example: $J(p_j, d_j)$: J1(1,1), J2(1,2) and J3(3,3.5) on two processor
- EDF (J3 misses) but the Optimal



# $P|ptmn|\Sigma U_j$

- In NPC