Approximation Algorithms II

Admin

- Assignment 8 feedback out; solutions on GLOW
- Assignment 9 is due tonight
- Assignment 10 (practice problems for final) will be released today
- Final review sessions/ office hours next week:
  - 2-3.30 pm on Monday (May 17)
  - **7-9 pm on Tuesday (May 18)**
  - 9-10.30 am Wed May 19 (in place of afternoon office hours)
- **Goal**: Come ask questions about the practice final or any thing from the past HWs/ lectures

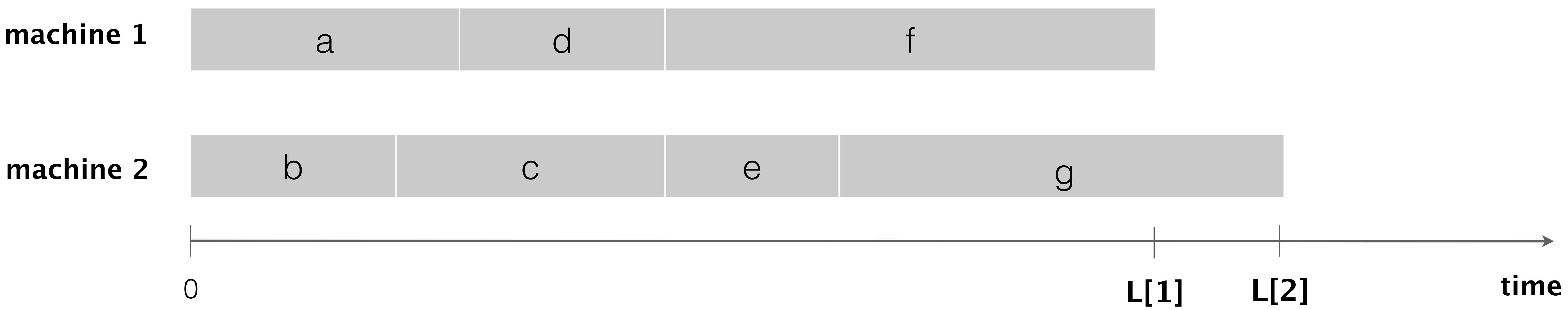
Final Logistics

- Final will be 24 hour take-home exam, open book, open GLOW
- Cumulative: cover all topics in course
- More focus on latter half:
  - dynamic programming, network flows, NP hardness reductions, randomized algorithms, approximation algorithms
- Reviewing problem sets 5-10 is good practice!
- Exam will be available on Gradescope from **Thurs May 20, 8.30 am**
- Start the exam whenever you are ready to take it
- All exams must be submitted by **May 28, 8.30 pm**

Load Balancing

Load Balancing

- **Input.** m identical machines and n jobs with processing times $t_1, \ldots, t_m$, where job j has processing time t_j (on any machine)
- Job j must run contiguously on one machine
- A machine can process at most one job at a time
- Let $S[i]$ be the subset of jobs assigned to machine i
- The **load of machine** i is $L[i] = \sum_{j \in S[i]} t_j$ (total processing time)



Load Balancing

- **Input.** m identical machines and n jobs with processing times $t_1, \ldots, t_m$, where job j has processing time t_j (on any machine)
- Let $S[i]$ be the subset of jobs assigned to machine i
- The **load of machine** i is $L[i] = \sum_{j \in S[i]} t_j$ (total processing time)
- The makespan of an algorithm is the maximum load on any machine $L = \max_i L[i]$
- Load balancing problem. Find an assignment of jobs to machines so as to minimize the makespan
  - Minimize the maximum load on any machine

Load Balancing is NP Hard

- **Decision Version.**
  - Given m identical machines and n jobs with processing times $t_1, \ldots, t_m$, where job j has processing time t_j (on any machine), and target L, does there exists an assignment of jobs to machines with make span at most L?
- **Claim.** Load balancing is NP hard even with $m = 2$ machines
- **Proof.** Reduction from Subset Sum
  - Given $S = x_1, \ldots, x_n$ and target T
  - Need to create jobs, assign processing times
  - Need a target makespan L

Load Balancing is NP Hard

- **Claim.** Load balancing is NP hard even with $m = 2$ machines
- **Proof.** Reduction from Subset Sum
  - Given $S = x_1, \ldots, x_n$ and target T
  - Create $n + 1$ jobs with processing times $x_1, \ldots, x_n, X - 2T$ where $X = \sum_{i=1}^{n} x_i$ and let target makespan be $X - T$
- $(\Rightarrow)$ Suppose $A \subseteq S$ is a subset of elements that sum to T
  - Then elements in $S - A$ sum to $X - T$
  - Assign jobs with processing times in S and job with processing time $X - 2T$ to machine 1, and rest to machine 2: makespan is $X - T$

Load Balancing is NP Hard

- **Claim.** Load balancing is NP hard even with $m = 2$ machines
- **Proof.** [Reduction from Subset Sum] Given $S = x_1, \ldots, x_n$ and target T
  - Create $n + 1$ jobs with processing times $x_1, \ldots, x_n, X - 2T$ where $X = \sum_{i=1}^{n} x_i$ and let target makespan be $X - T$
- $(\Leftarrow)$ Suppose the makespan is $X - T$, since the total processing time is $2X - 2T$, that must be split evenly across the machine
- That is, load of each machine is $X - T$
- Wlog say job with processing time $X - 2T$ is on machine 1, then the processing time of remaining jobs on that machine must sum to T

Load Balancing: Greedy

- Go through the jobs one by one
- Assign each job to the machine with the smallest load so far

GREEDY-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

$i \leftarrow \operatorname{argmin}_k L[k].$ ← 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: Greedy

- Can implement greedy in $O(n \log m)$ time

GREEDY-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 $S[1], S[2], \ldots, S[m].$

Load Balancing: Greedy

- How good is greedy?
- That is, how good is the makespan of the assignment returned by greedy?

GREEDY-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 $S[1], S[2], \ldots, S[m].$

Load Balancing: Greedy Analysis

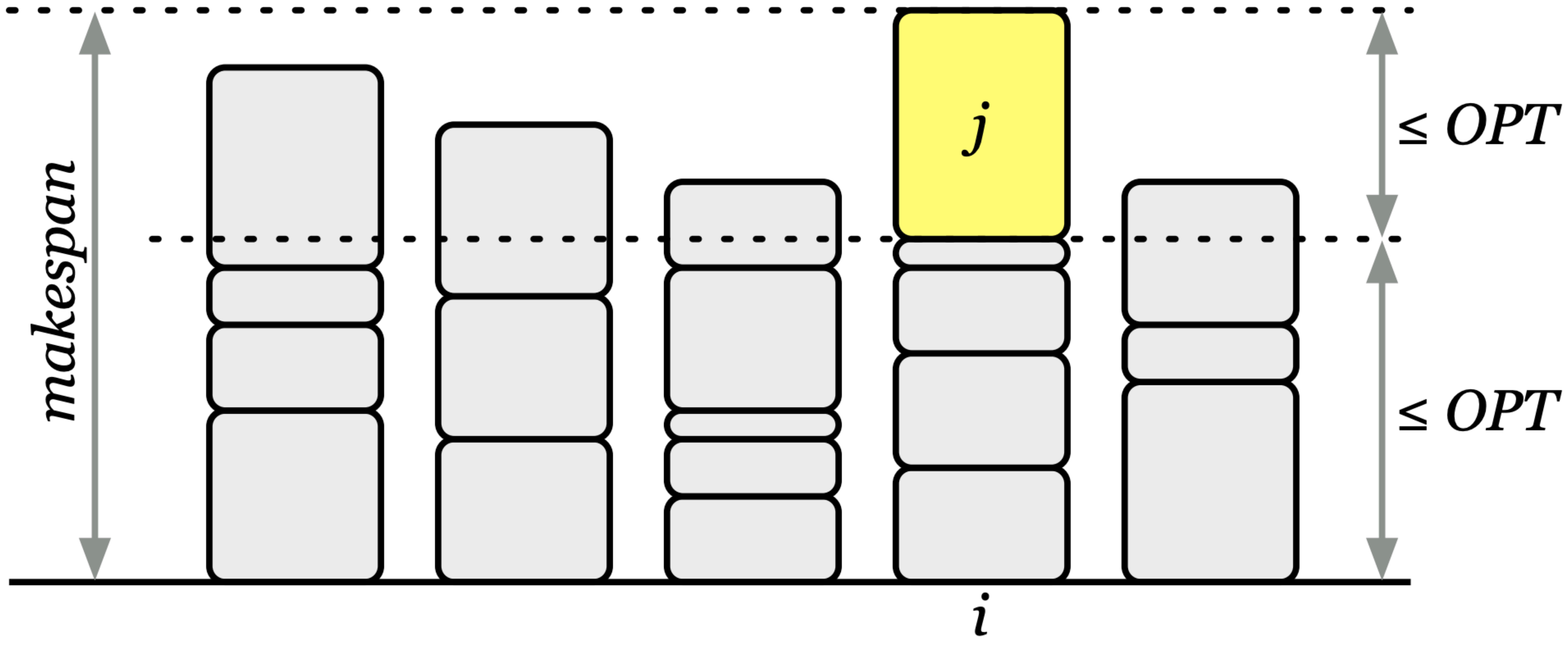
- **Claim.** Greedy algorithm is a 2-approximation.
- To show this, we need to show greedy solution never more than a factor two worse than the optimal
- **Challenge.** We don't know the optimal solution (finding it is NP hard)
- Steps to show approximation factor:
    - Lower bound the cost of optimal solution
    - A good enough lower bound can help show that our algorithm cannot be too much worse than the optimal
- In our problem, what are some lower bounds on the makespan of even an optimal algorithm?

Load Balancing: Greedy Analysis

- Let OPT denote the optimal makespan
- **Lemma**. OPT $\geq \max_j t_j$ (max processing time among all jobs)
- Proof. Some machine must process the most time-consuming job.
- Any other lower bounds?
- **Lemma**. OPT $\geq \frac{1}{m} \sum_j t_j$
- **Proof**. The total processing time is $\sum_j t_j$
  - Some machine must do a $1/m$ fraction of the total work

Greedy is a 2-Approximation

- **Proof.** Consider load $L[i]$ of bottleneck machine i ⟵ machine that ends up with highest load
- Let j be the last scheduled job on machine i
- When job j was assigned to machine i, i must have had the smallest load
- That is, $L[i] - t_j \leq L[k] \quad \forall 1 \leq k \leq m$



Greedy is a 2-Approximation

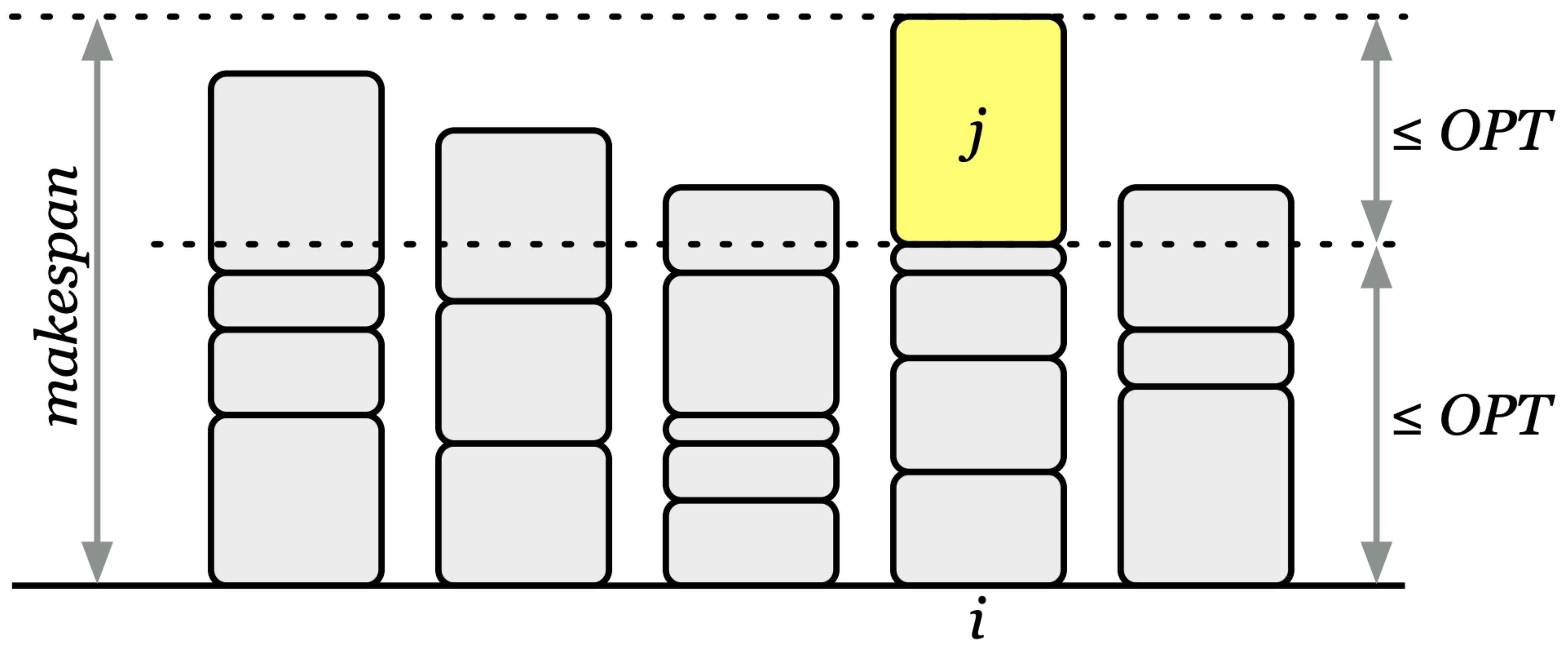
- **Proof.** Consider load $L[i]$ of bottleneck machine i
- Let j be the last scheduled job on machine i
- When job j was assigned to machine i, i must have had the smallest load
- That is, $L[i] - t_j \leq L[k] \qquad \forall 1 \leq k \leq m$
- Summing over all k and diving by m

$$
\begin{aligned}
L[i] - t_j &\leq \frac{1}{m} \sum_k L[k] \\
&\leq \frac{1}{m} \sum_{j'} t_{j'} \\
&\leq \text{OPT}
\end{aligned}
$$

Greedy is a 2-Approximation

- **Proof**.
- Consider load $L(i)$ of bottleneck machine i
- $L[i] - t_j \leq$ OPT
- We know that $t_j \leq$ OPT
- Thus, $L = L[i] \leq \text{OPT} + t_j$

  $\leq 2\text{OPT}$ ∎



Greedy is a 2-Approximation

- Is our analysis tight?
  - Close to it.
- Consider $m(m-1)$ jobs of length 1 and 1 job of length m
- How would greedy schedule these jobs?
  - Greedy will evenly divide the first $m(m-1)$ jobs among m machines, will place the final long job on any one machine
  - Makespan: $m - 1 + m = 2m - 1$
- How would optimal schedule it?
  - Give the long job to one machine, the rest split the other small jobs with a makespan m
- Ratio: $(2m-1)/m \approx 2$

Greedy is Online

- Notice that our greedy algorithm is an online algorithm
- Assigns jobs to machines in the order they arrive
  - Does not depend on future jobs
- Can we do better, if we assume all jobs are available at start time?
- **Offline.** Slight modification of greedy gets better approximation!

Improving on Online Greedy

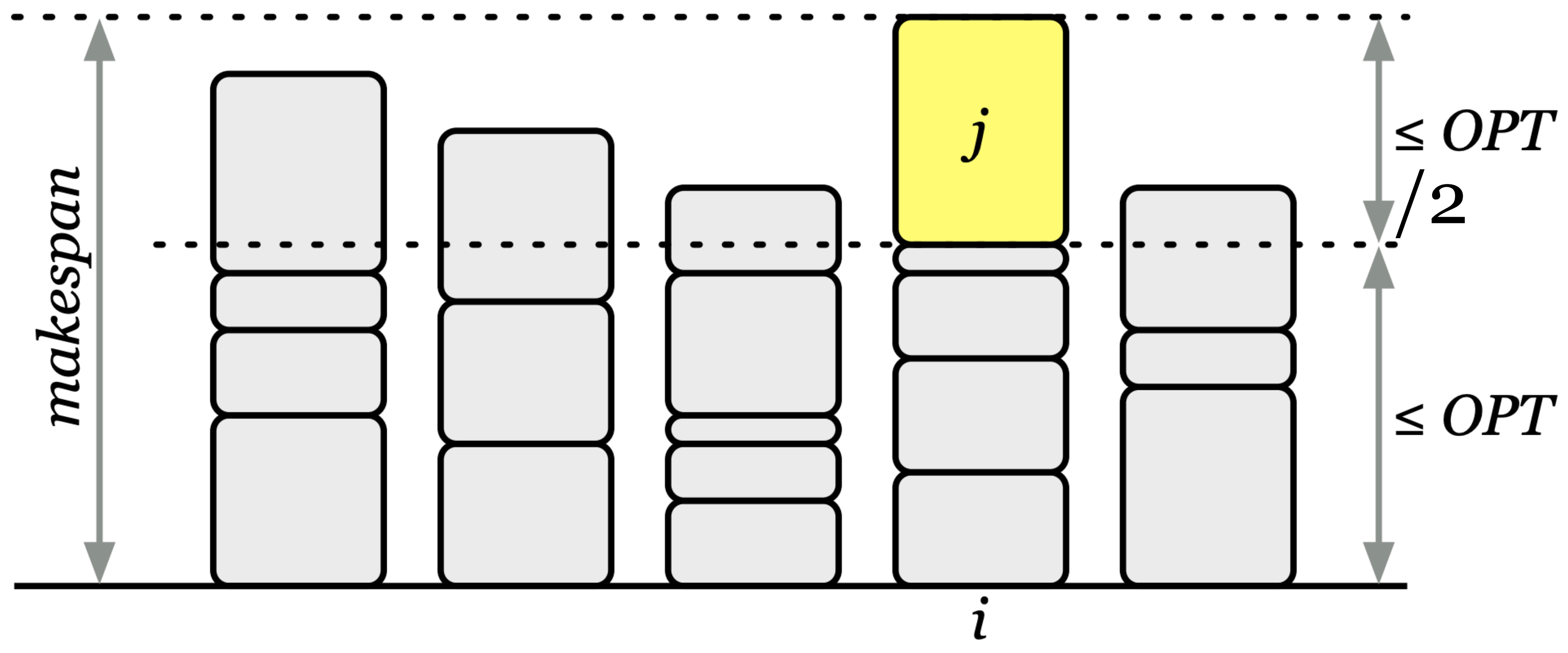
- Worst case for greedy: spreading jobs out evenly when a giant job at the end messed things up
- What can we do to avoid this?
  - Idea: deal with larger jobs first
  - Small jobs can only hurt so much
- Turns out this improves our approximation factor
- **Longest-processing-time (LPT) first**. Sort n jobs in decreasing order of processing times; then run the greedy algorithm on them
- **Claim.** LPT has a makespan at most $1.5 \cdot \text{OPT}$
- **Observation.** If we have fewer than m jobs, then the greedy solution is clearly optimal (as it puts each job on its own machine)

LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot$ OPT
- **Observation.**
  - If we have fewer than m jobs, then the greedy solution is clearly optimal (as it puts each job on its own machine)
- **Claim.** If more than m jobs then, OPT $\geq 2 \cdot t_{m+1}$
- **Proof.** Consider the first $m+1$ jobs in sorted order.
  - They each take at least t_{m+1} time
  - $m+1$ jobs and m machines, there must be a machine with at least two jobs
  - Thus the optimal makespan OPT $\geq 2 \cdot t_{m+1}$

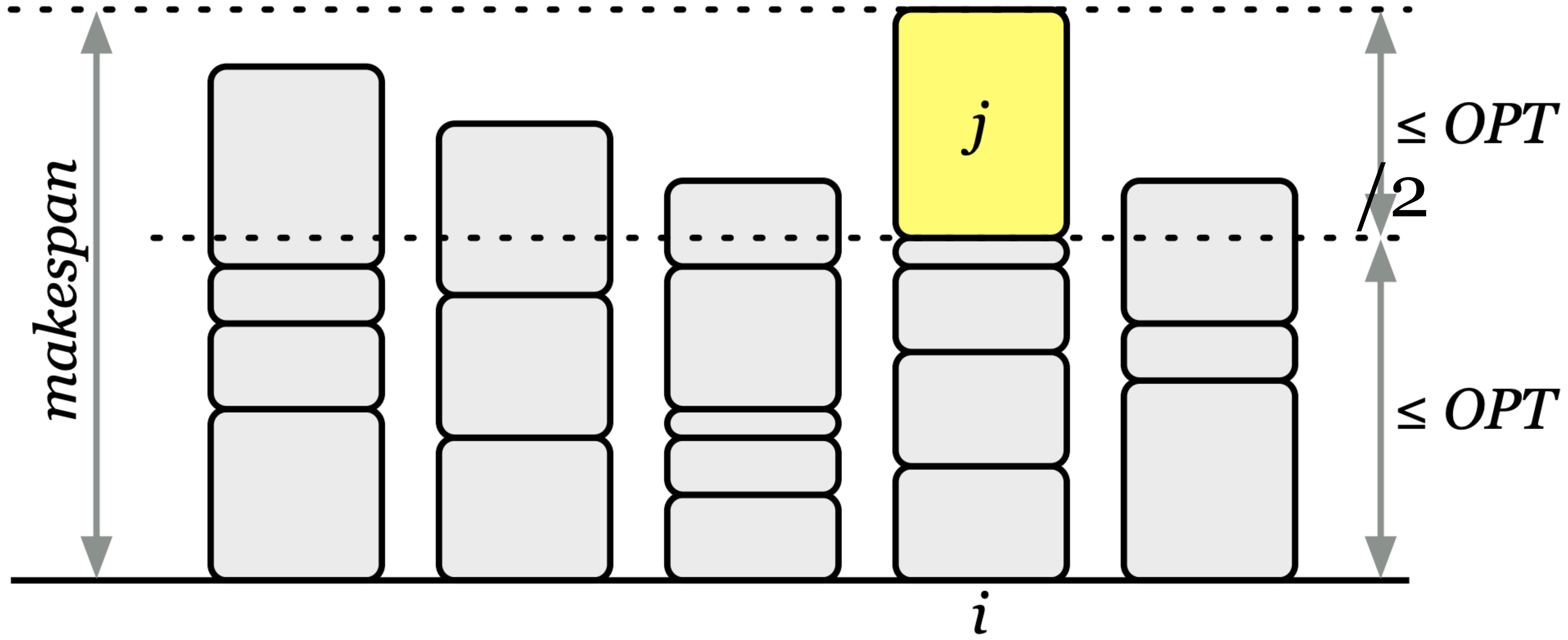
LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot$ OPT
- **Proof**. (Similar to our original proof.)
- Consider the machine M_i that has the maximum load
- If M_i has a single job, then our algorithm is optimal
- Suppose M_i has at least two jobs and let t_j be the last job assigned to the machine, note that $j \geq m+1$ (why?)
- Thus, $t_j \leq t_{m+1} \leq \frac{1}{2}$OPT



LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot$ OPT
- **Proof.** As before, consider the machine M_i that has the maximum load
- If M_i has a single job, then our algorithm is optimal
- Suppose M_i has at least two jobs and let t_j be the last job assigned to the machine, note that $j \geq m + 1$ (why?)
- Thus, $t_j \leq t_{m+1} \leq \frac{1}{2}$OPT
- $L[i] - t_j \leq$ OPT
- $L[i] \leq \frac{3}{2}$OPT ∎



Is our 1.5-Approximation tight?

- **Question.** Is out $3/2$-approximation analysis tight?
  - Turns out, no
- **Theorem [Graham 1969].** LPT-first is a $4/3$-approximation.
  - Proof via a more sophisticated analysis of the same algorithm
- **Question.** Is the $4/3$-approximation analysis tight?
  - Pretty much.
- Example
  - m machines, $n = 2m + 1$ jobs
  - 2 jobs each of length $m, m + 1, \ldots, 2m - 1$ + one job of length m
  - Approximation ratio $= (4m - 1)/3m \approx 4/3$

Load Balancing: Where We Are

- Long series of improvements
- Polynomial time algorithm for *any* constant approximation [Hochbaum Shmoys 87]
- Specifically: $(1+\epsilon)$ approximation in $O\left((n/\epsilon)^{1/\epsilon^2}\right)$ time
- **PTAS:** Polynomial time approximation scheme
- For any desired constant-factor approximation ϵ, there exists a polynomial-time algorithm

Acknowledgments

- Some of the material in these slides are taken from
    - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
    - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
    - Lecture slides: https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/