Approximation Algorithms

02282

Inge Li Gørtz

Approximation algorithms

- Fast. Cheap. Reliable. Choose two.
- NP-hard problems: choose 2 of
    - optimal
    - polynomial time
    - all instances

- **Approximation algorithms.** Trade-off between time and quality.

- Let A(I) denote the value returned by algorithm A on instance I. Algorithm A is an *α-approximation algorithm* if for any instance I of the optimization problem:
    - A runs in polynomial time
    - A returns a valid solution
    - $A(I) \leq \alpha \cdot OPT$, where $\alpha \geq 1$, for minimization problems
    - $A(I) \geq \alpha \cdot OPT$, where $\alpha \leq 1$, for maximization problems

Examples

- **Acyclic Graph** Given a directed graph G=(V,E), pick a maximum cardinality set of edges from E such that the resulting graph is acyclic.
  - Give a 1/2-approximation algorithm for this problem.

- **Minimum Maximal Matching**
  - A matching in a graph G=(V,E) is a subset of edges $M \subseteq E$, such that no two edges in M share an endpoint.
  - A maximal matching is a matching that cannot be extended, i.e., it is not possible to add an edge from $E \setminus M$ to M without violating the constraint.
  - Design a 2-approximation algorithm for finding a minimum cardinality maximal matching in an undirected graph.

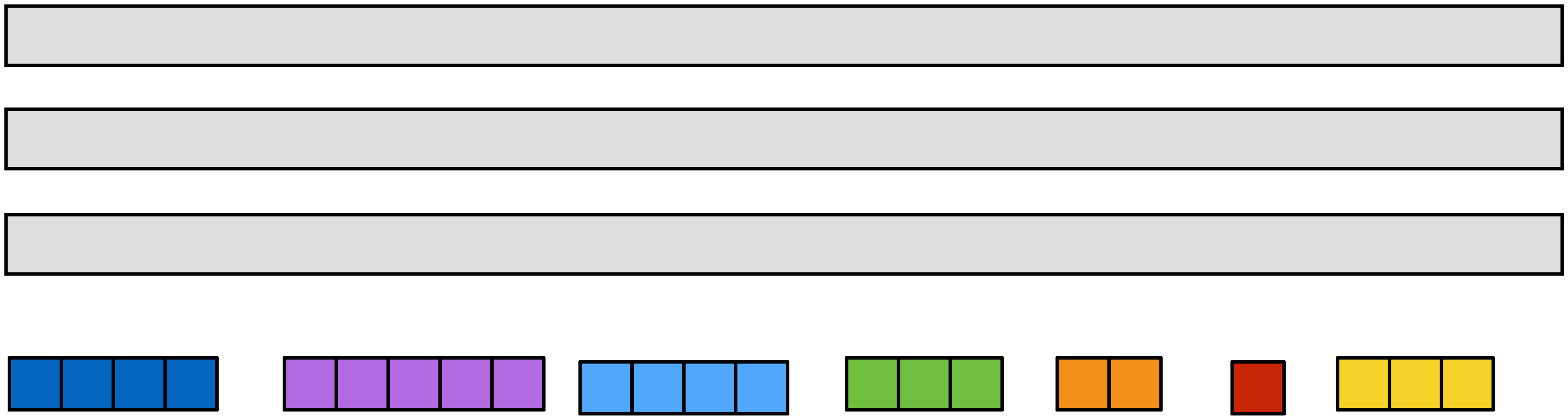
Examples

- **Acyclic Graph** Given a directed graph G=(V,E), pick a maximum cardinality set of edges from E such that the resulting graph is acyclic.
  - Give a 1/2-approximation algorithm for this problem.
  - Lower bound - what is the best we can hope for?
  - Arbitrarily number the vertices and pick the bigger of the two sets, the forward going edges and the backward going edges.

- **Minimum Maximal Matching**
  - A matching in a graph G=(V,E) is a subset of edges M \subseteq E, such that no two edges in M share an endpoint.
  - A maximal matching is a matching that cannot be extended, i.e., it is not possible to add an edge from E\setminus M to M without violating the constraint.
  - Design a 2-approximation algorithm for finding a minimum cardinality maximal matching in an undirected graph.
  - Lower bound: Any maximal matching is at least half the maximum maximal matching. Why?

Load balancing

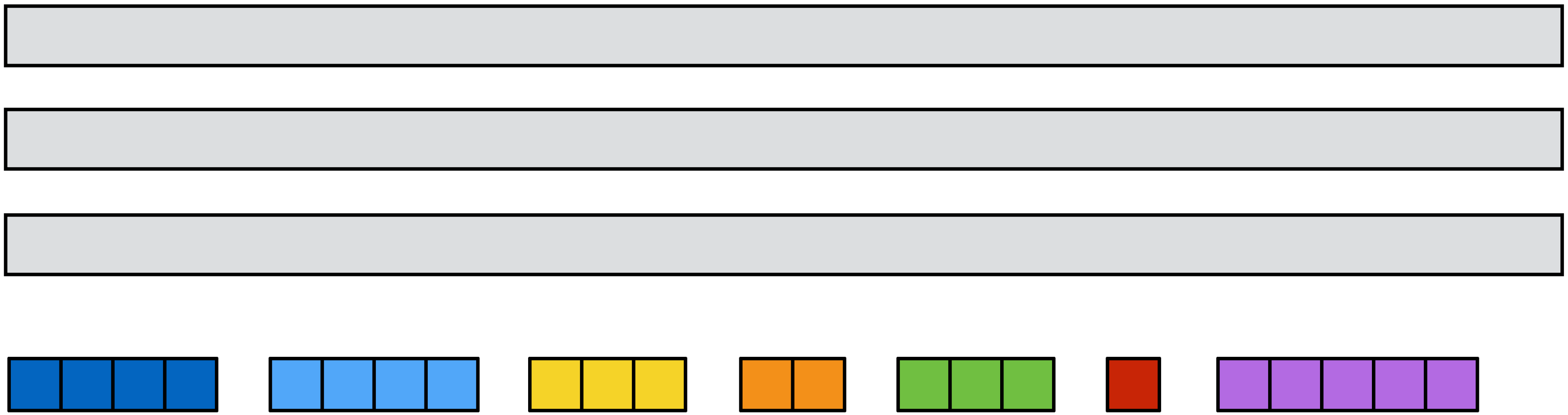
Scheduling on identical parallel machines



- n jobs to be scheduled on m identical machines.
- Each job has a processing time t_j.
- Once a job has begun processing it must be completed.
- T_j: Load of machine j.
- Goal. Schedule all jobs so as to *minimize the maximum load (makespan)*:

$$\text{minimize } T = \max_{i=1\ldots n} T_j$$

Simple greedy (list scheduling)



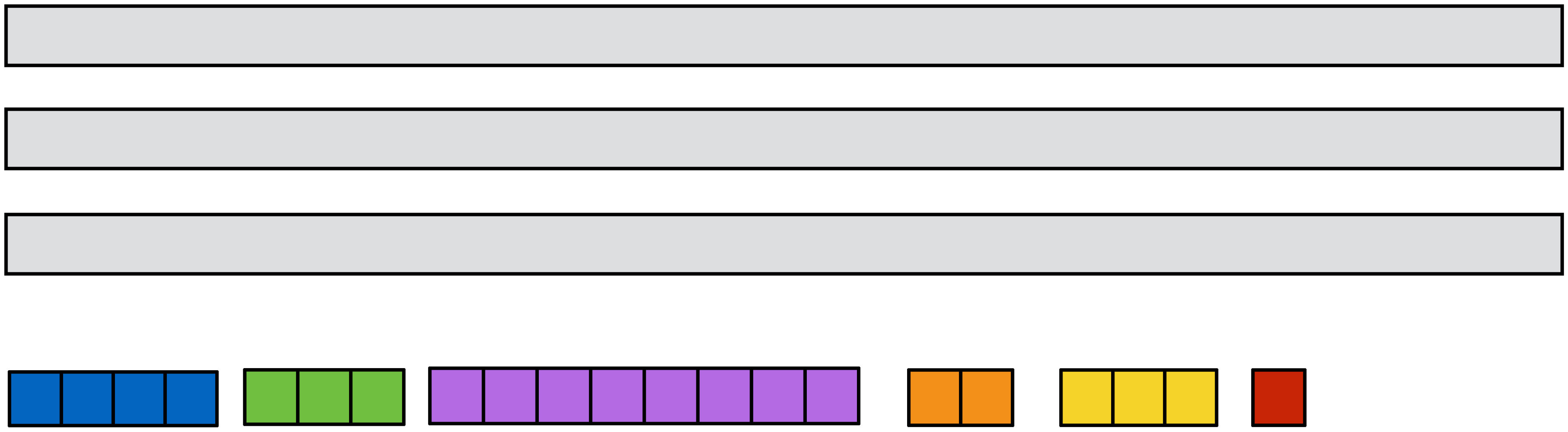
- *Simple greedy.* Process jobs in any order. Assign next job on list to machine with smallest current load.

- The greedy algorithm above is a 2-approximation algorithm:
  - polynomial time ✓
  - valid solution ✓
  - factor 2

Simple greedy (list scheduling)

Simple greedy (list scheduling)

Simple greedy (list scheduling)



Approximation factor

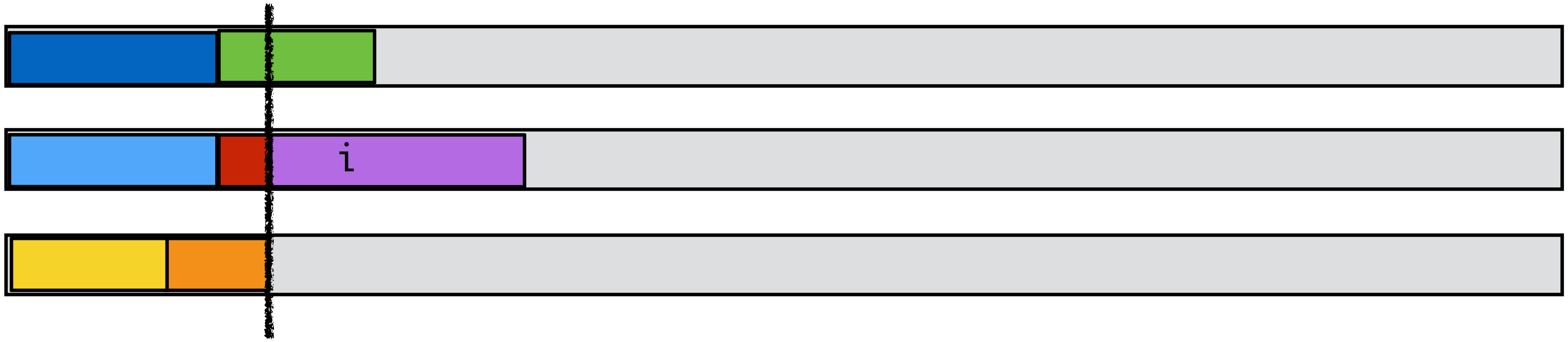
- Lower bounds:
    - Each job must be processed:

$$T^* \geq \max_j t_j$$

    - There is a machine that is assigned at least average load:

$$T^* \geq \frac{1}{m} \sum_j t_j$$

Approximation factor

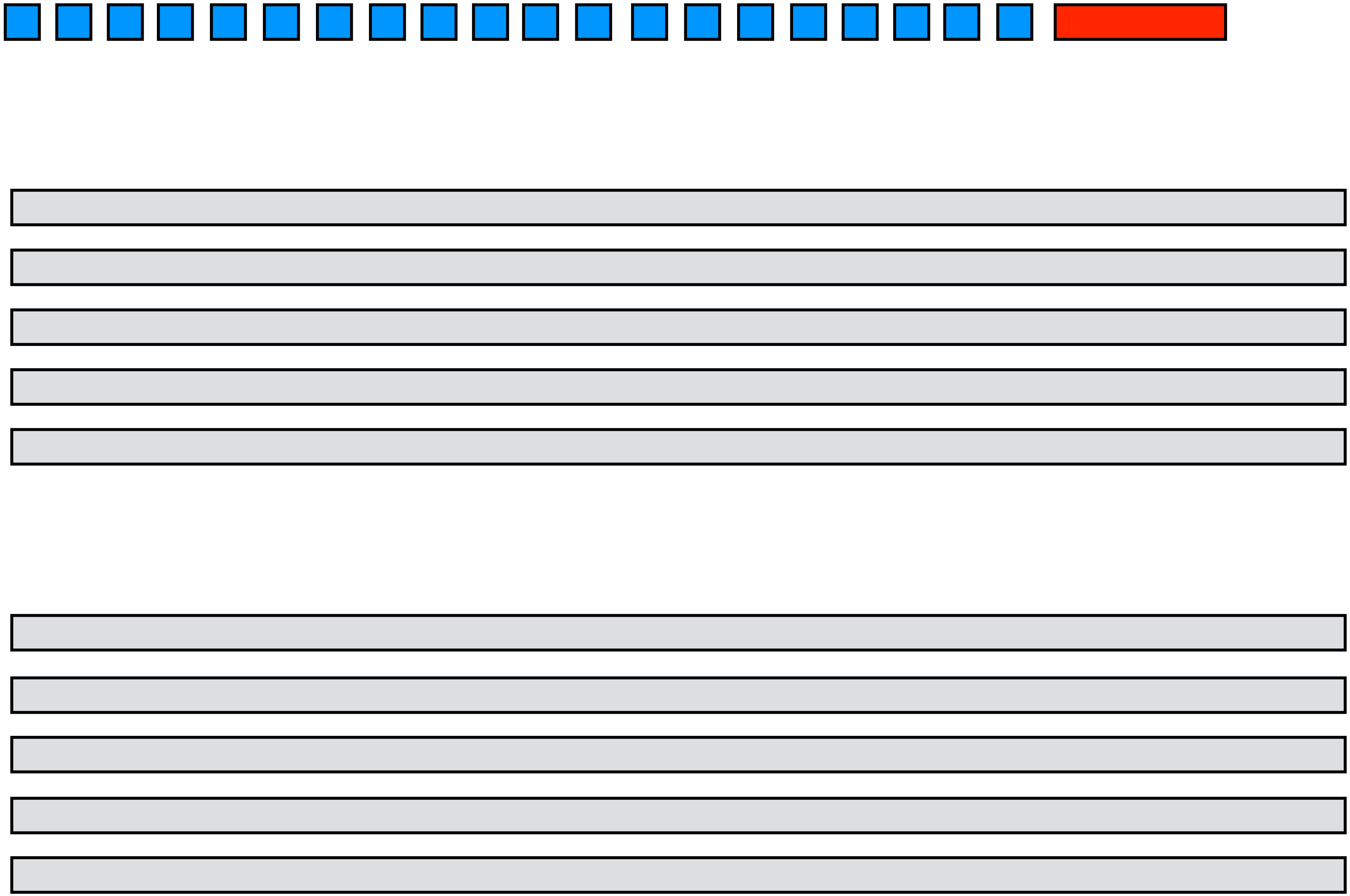


- i: job finishes last.
- All other machines busy until start time s of i. ($s = T_i - t_i$)
- Partition schedule into before and after s.
- After $\leq T^*$.
- Before:
  - All machines busy => total amount of work = $m \cdot s$:

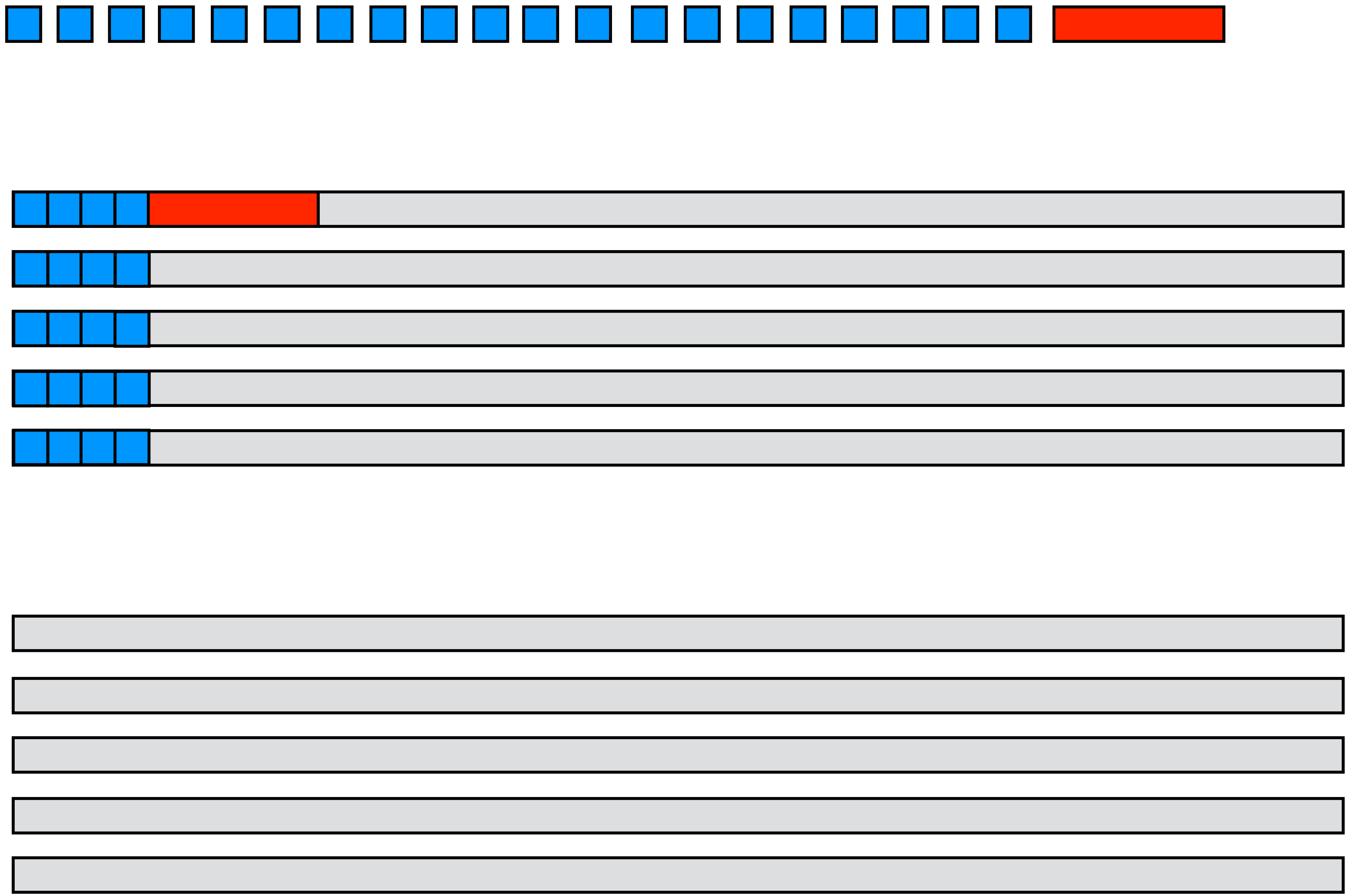
$$m \cdot s \leq \sum_j t_j \qquad \Rightarrow s \leq \frac{1}{m} \sum_j t_j \leq T^*$$

  - Length of schedule = $s + t_i \leq T^* + T^* = 2T^*$.

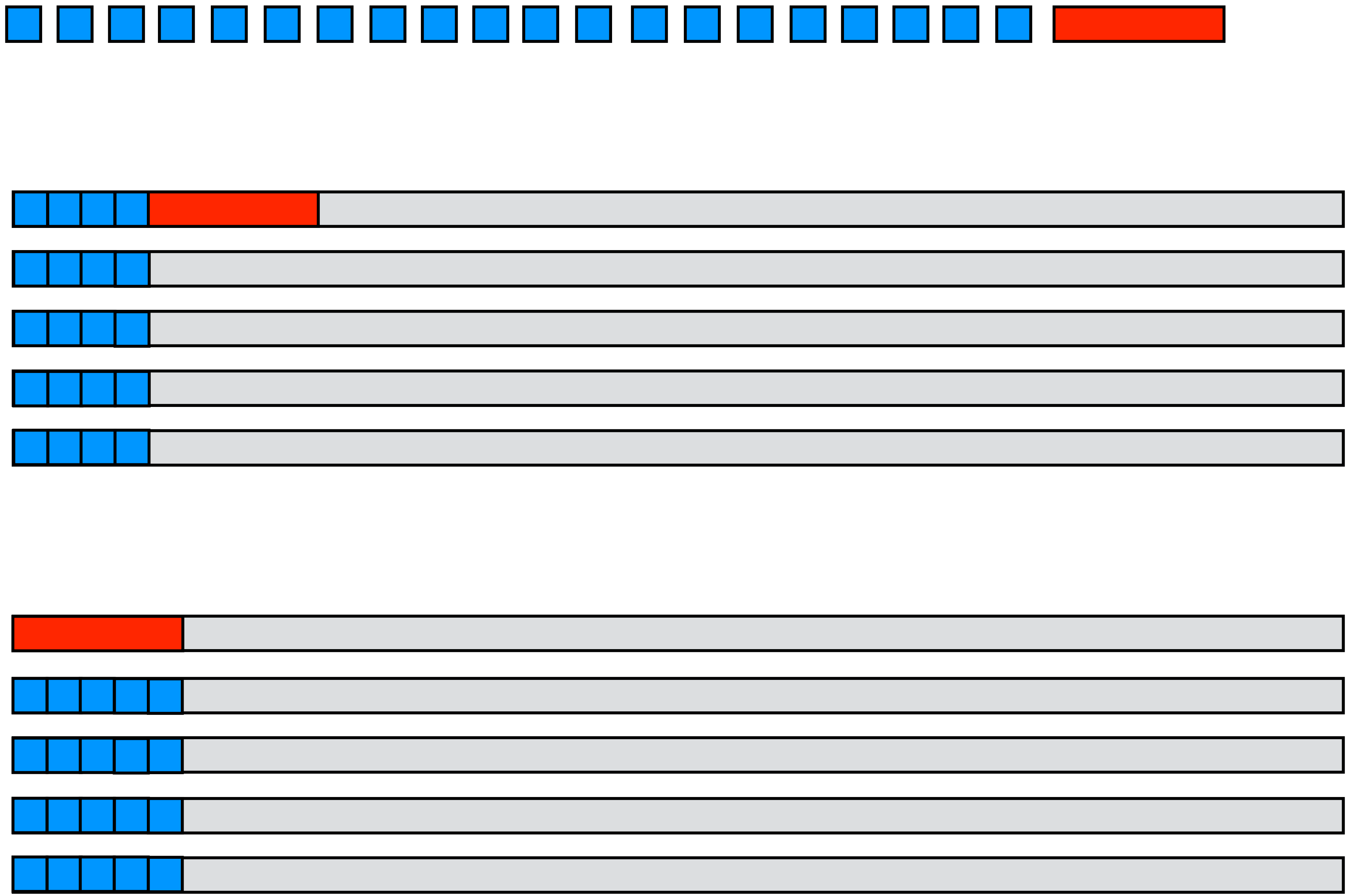
Lower bound



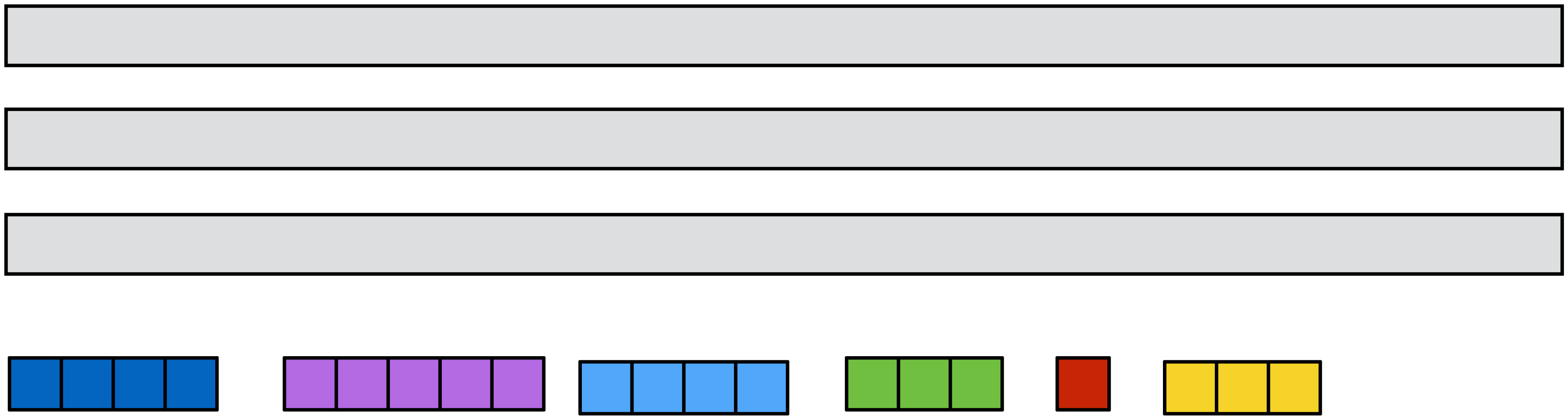
Lower bound



Lower bound

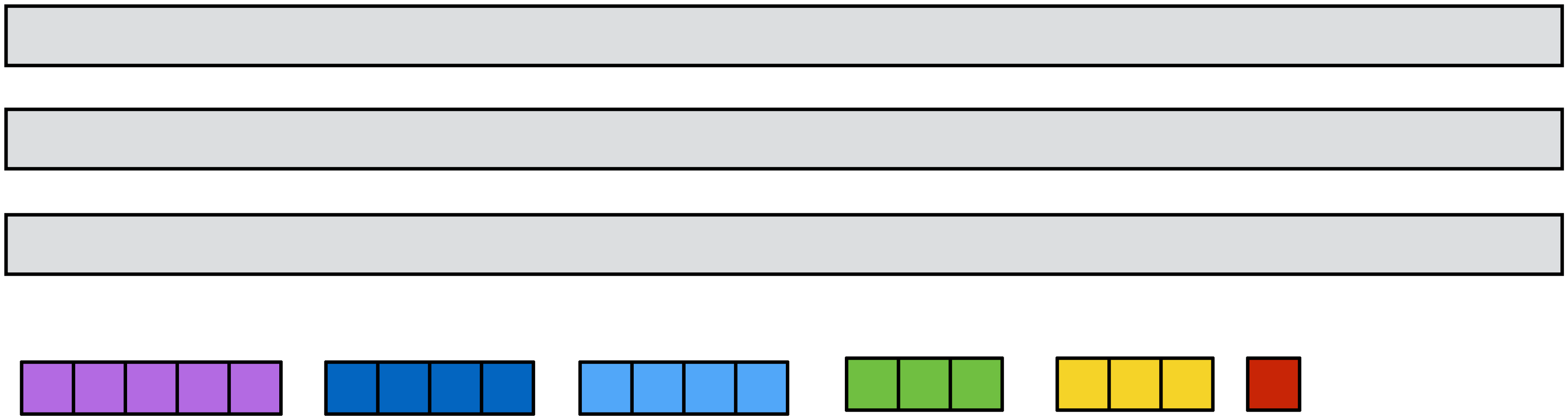


Longest processing time rule



- *Longest processing time rule (LPT).* Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.

Longest processing time rule



- *Longest processing time rule (LPT).* Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.
- LPT is a is a 3/2-approximation algorithm:
  - polynomial time ✓
  - valid solution ✓
  - factor 3/2

Longest processing time rule: factor 3/2

- **Longest processing time rule (LPT).** Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.
- Assume $t_1 \geq \ldots. \geq t_n$.
- If $n \leq m$ then optimal.
- Lower bound: If $n > m$ then $T^* \geq 2t_{m+1}$.
- Factor 3/2:
  - Before $\leq T^*$
  - After: i job that finishes last.
    - $t_i \leq t_{m+1} \leq T^*/2$.
  - $T \leq T^* + T^*/2 \leq 3/2\ T^*$.
- Tight?

Longest processing time rule: factor 4/3

- **Longest processing time rule (LPT).** Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.
- Assume $t_1 \geq \ldots. \geq t_n$.
- Assume wlog that smallest job finishes last.
- If $t_n \leq T^*/3$ then $T \leq 4/3\ T^*$.
- If $t_n > T^*/3$ then each machine can process at most 2 jobs in OPT.
- **Lemma.** *For any input where the processing time of each job is more than a third of the optimal makespan, LPT computes an optimal schedule.*
- **Theorem.** *LPT is a 4/3-approximation algorithm.*

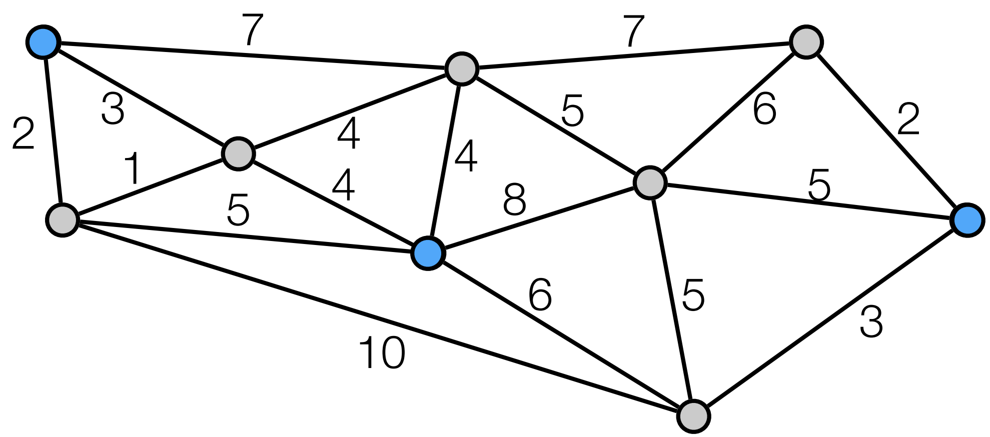
k-center

The k-center problem

- **Input**. An integer k and a set of sites S with distance d(i,j) between each pair of sites $i,j \in S$.
- d is a metric:
  - dist(i,i) = 0
  - dist(i,j) = dist(j,i)
  - dist(i,l) ≤ dist(i,j) + dist(j,l)
- **Goal.** Choose a set $C \subseteq S$, $|C| = k$, of k centers so as to minimize the maximum distance of a site to its closest center.

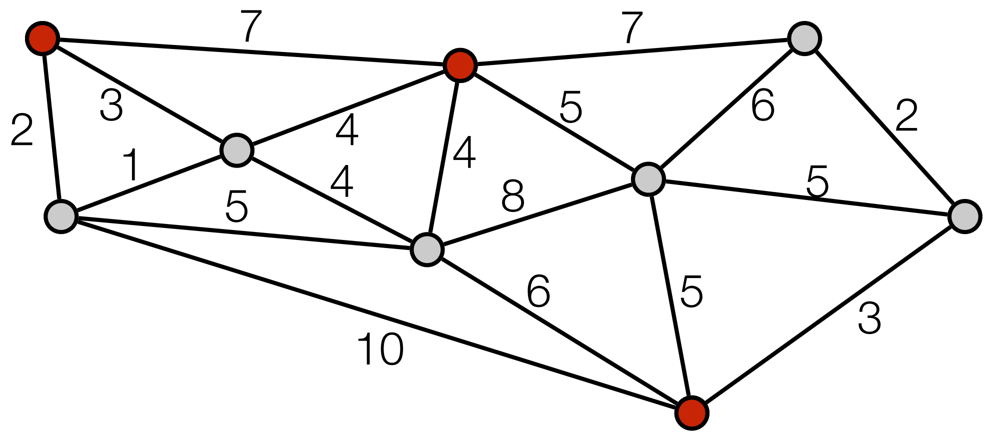
$$C = \text{argmin}_{C \subseteq V, |C|=k} \max_{i \in V} \text{dist}(i,C)$$

- **Covering radius.** Maximum distance of a site to its closest center.



k-center: Greedy algorithm

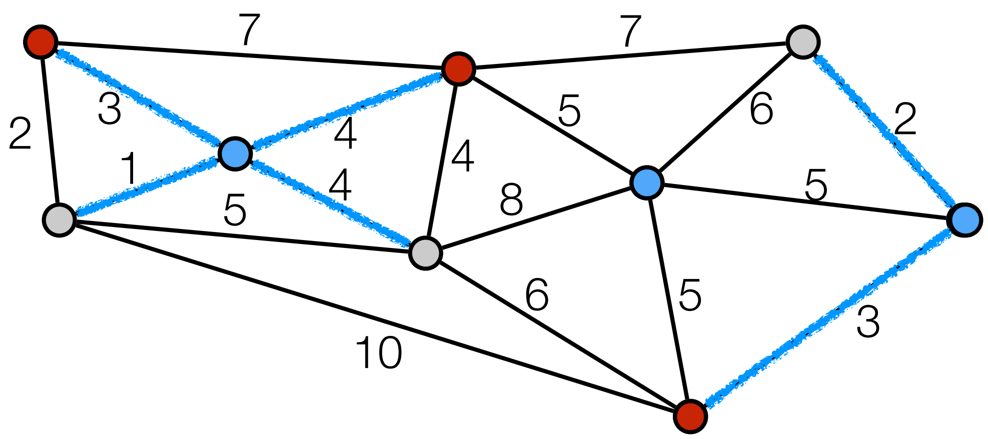
- Greedy algorithm.
  - Pick arbitrary i in S.
  - Set C = {i}
  - while |C| < k do
    - Find vertex j farthest away from any cluster center in C
    - Add j to C
  - Return C



- Greedy is a 2-approximation algorithm:
  - polynomial time ✓
  - valid solution ✓
  - factor 2

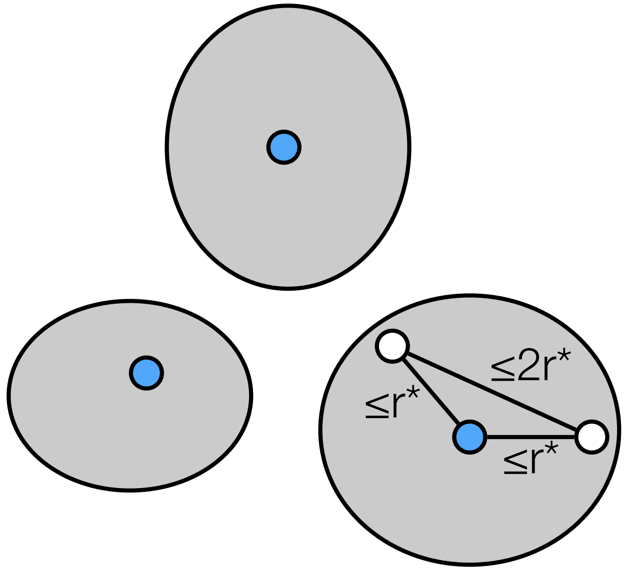
k-center analysis: optimal clusters

- Optimal clusters: each vertex assigned to its closest optimal center.

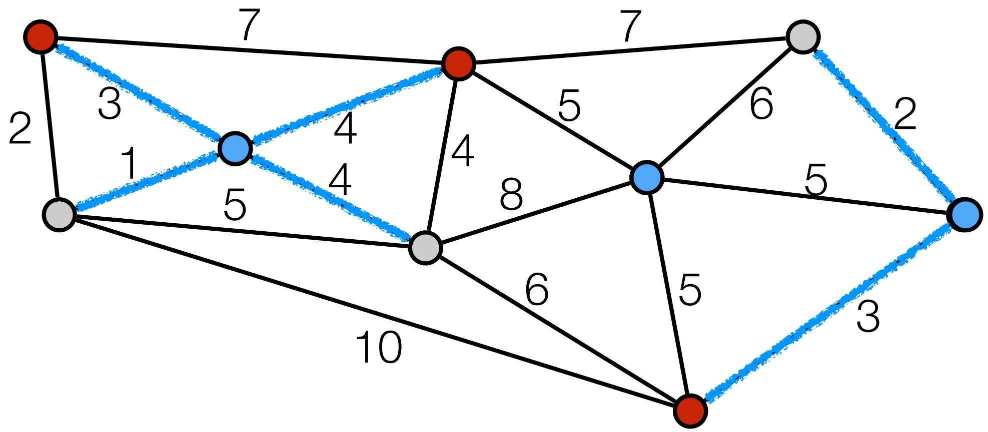


k-center analysis

- r* optimal radius.
- **Claim:** Two vertices in same optimal cluster has distance at most 2r* to each other.



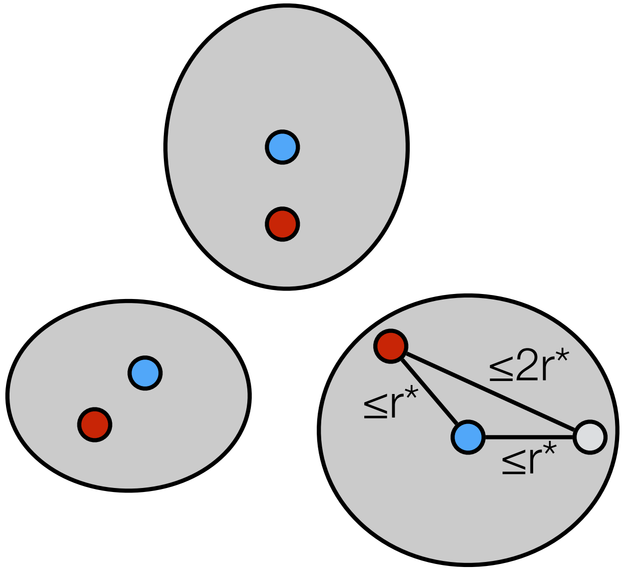
k-center



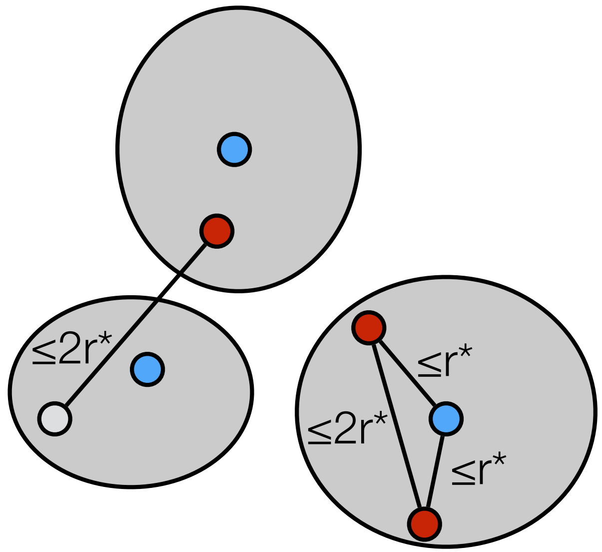
k-center: analysis greedy algorithm

- r* optimal radius.
- Show all vertices within distance 2r* from a center.
- Consider optimal clusters. 2 cases.

   1. *Algorithm picked one center in each optimal cluster*
      - distance from any vertex to its closest center ≤ 2r*.

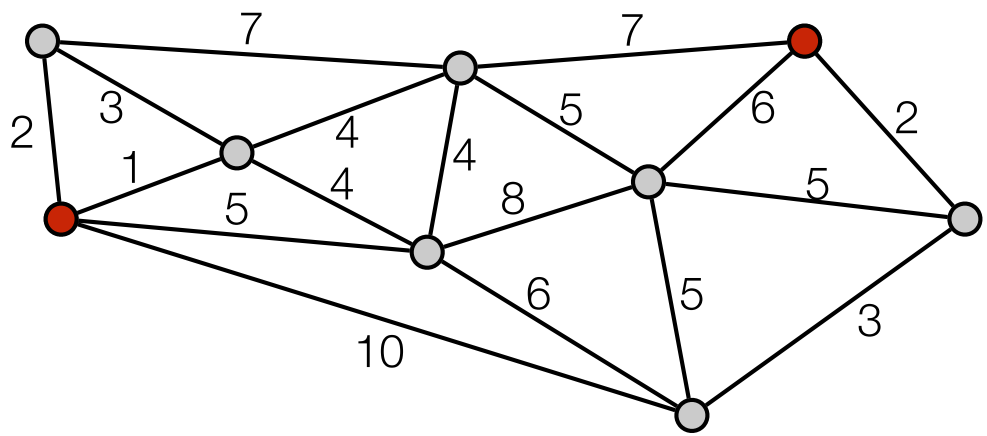


   2. *Some optimal cluster does not have a center.*
      - Some cluster have more than one center.
      - Distance between these two centers ≤ 2r*.
      - When second center in same cluster picked it was the vertex farthest away from any center.
      - Distance from any vertex to its closest center at most 2r*.



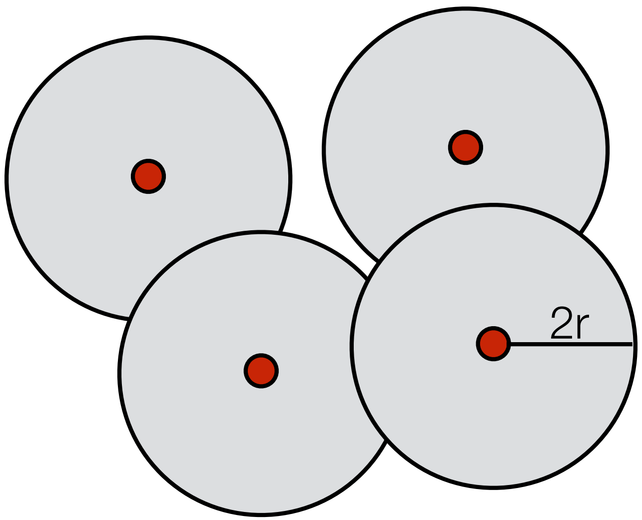
Bottleneck algorithm

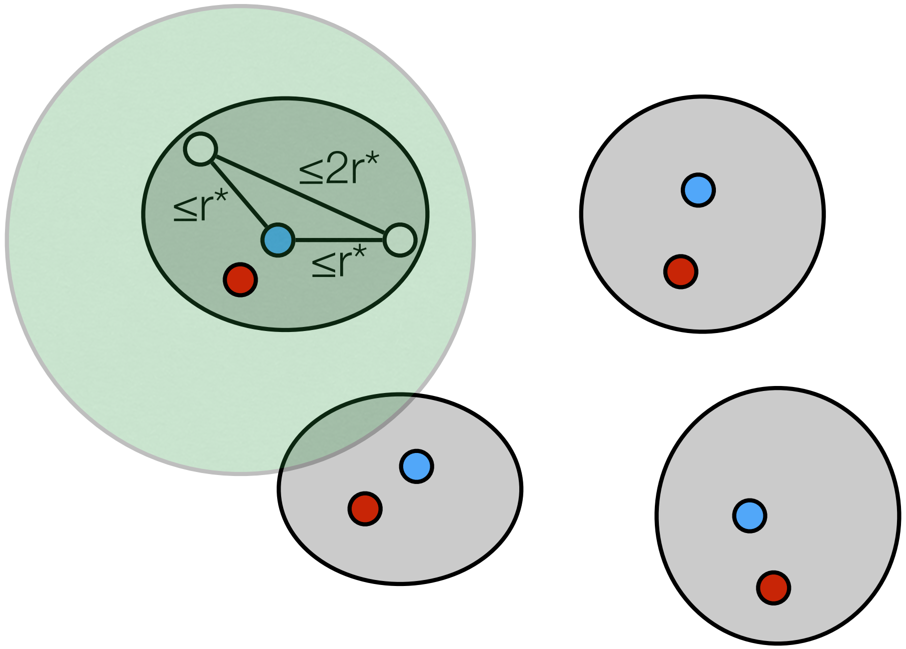
- Assume we know the optimum covering radius r.
- Bottleneck algorithm.
  - Set R := S and C := Ø.
  - while R ≠ Ø do
    - Pick arbitrary i in R.
    - Add j to C
    - Remove all vertices with d(j,v) ≤ 2r from R.
  - Return C
- Example: k= 3. r = 4.



Analysis bottleneck algorithm

- r^* optimal radius.
- Covering radius is at most $2r = 2r^*$.
- Show that we cannot pick more than k centers:
  - We can pick at most one in each optimal cluster:
    - Distance between two nodes in same optimal cluster $\leq 2r^*$.
    - When we pick a center in a optimal cluster all nodes in same optimal cluster is removed.





Analysis bottleneck algorithm

- r* optimal radius.
- Can use algorithm to “guess” r* (at most n^2 values).
- If algorithm picked more than k centers then r* > r.
    - If algorithm picked more than k centers then it picked more than one in some optimal cluster.
    - Distance between two nodes in same optimal cluster ≤ 2r.*
    - If more than one in some optimal cluster then 2r < 2r*.