Complexity & Algorithms, Spring '26

Approximation:
Load Balancing
+ Clustering

Recap

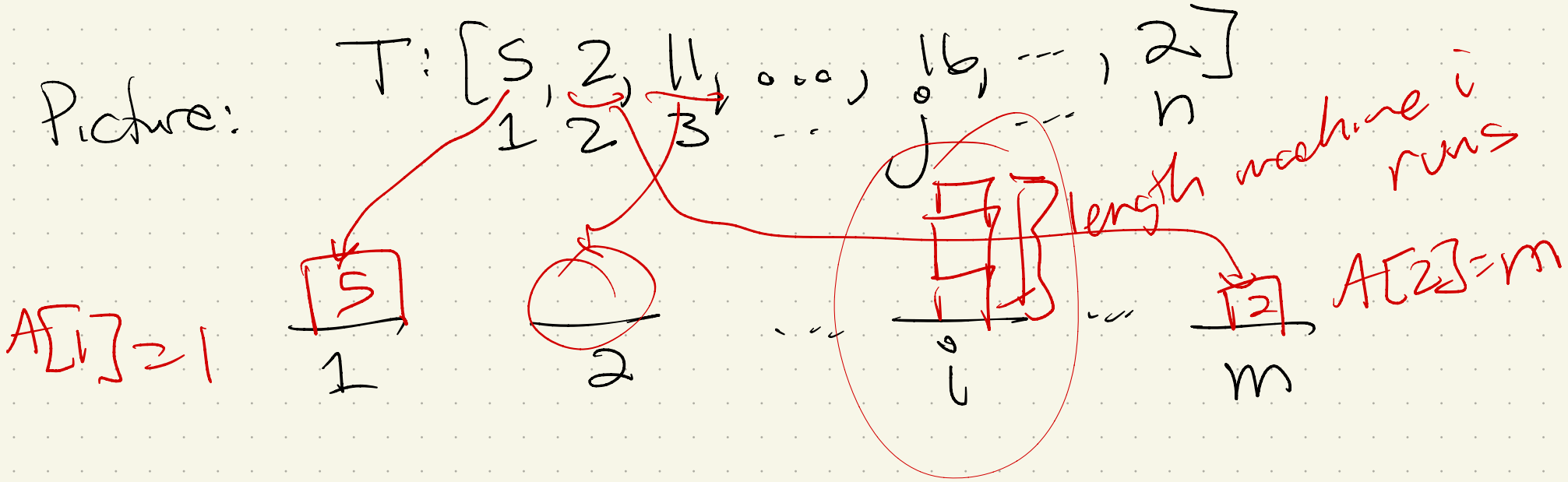
- Posted next two readings
  - ↳ NP-Hardness
- HW2 — signups are on Canvas

Load Balancing

- n jobs, each with a running time $T[1..n]$
- m machines available on which to run them

Goal: Compute an assignment $A[1..n]$ where job j gets assigned to some machine $i \in [1..m]$, i.e. $A[j] = i$

Picture:



Natural Goal:

Finish as early as possible!

Makespan: max time any machine is running jobs:

$$\text{makespan}(A) = \max_{i} \left(\sum_{j:\, A[j]=i} T[j] \right)$$

i: machines; $T[j]$: runtime

Goal: $\min_{A}$ (makespan (A))

Bad news: NP-Hard

Approximating

What seems a natural strategy?

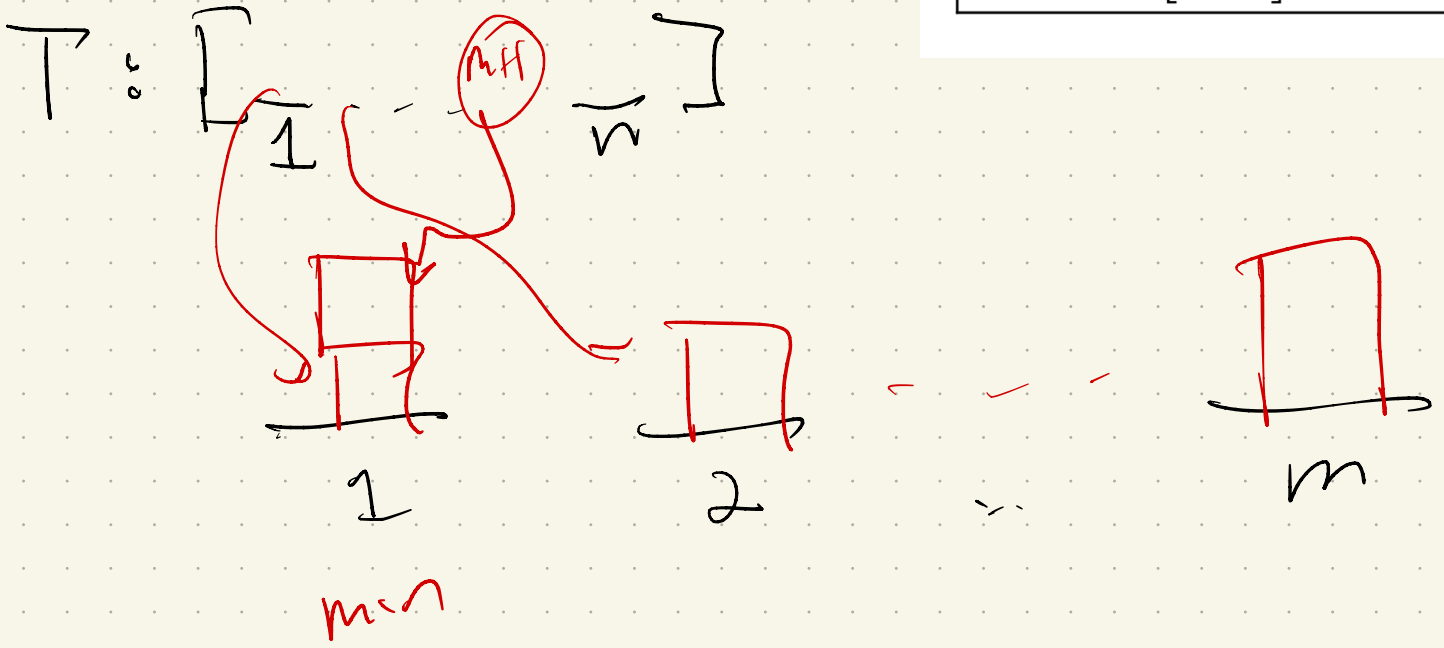
Be greedy!

Think of these one at a time:

```
GreedyLoadBalance(T[1..n], m):
  for i ← 1 to m
      Total[i] ← 0
  for j ← 1 to n
      mini ← arg min_i Total[i]
      A[j] ← mini
      Total[mini] ← Total[mini] + T[j]
  return A[1..m]
```

loop through jobs

find current shortest machine & go there



Claim: Greedy makespan $\leq 2 \cdot OPT$

we don't know this!

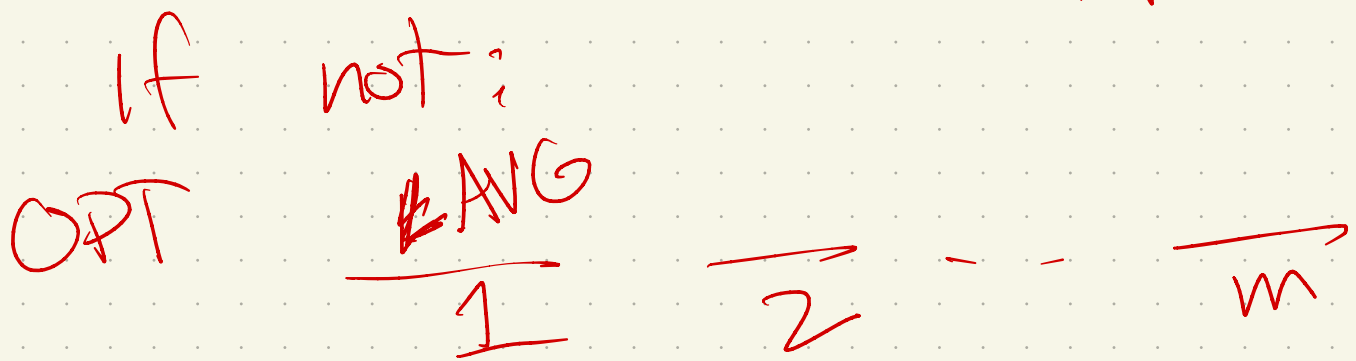
Proof: 2 observations:

① $OPT \geq$ longest job $= \max_j T[j]$

if less, not running job j

② $OPT \geq$ average of all jobs $= \frac{1}{m}\left(\sum_j T[j]\right)$

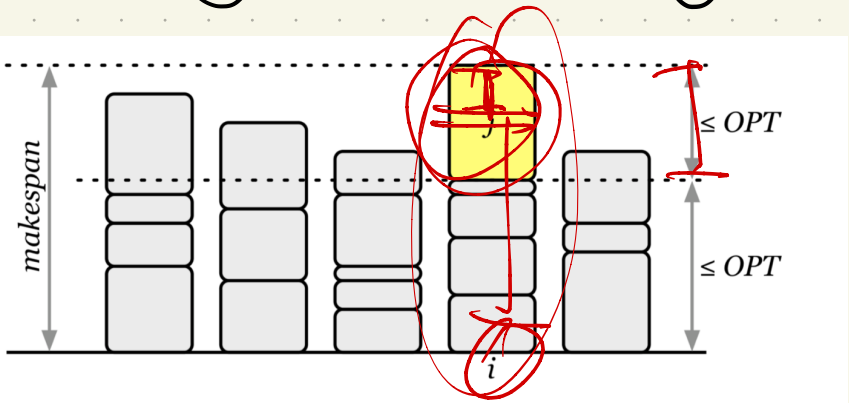
if not:



Now consider machine w/ largest makespan in greedy alg → machine i.

Note: Cost(greedy) = $\sum_{\text{all } A[j]=i} T[j]$

Let j be last job assigned to machine i.



Proof that GREEDYLOADBALANCE is a 2-approximation algorithm

Use ①: $OPT \geq \max T[k] \geq T[j]$

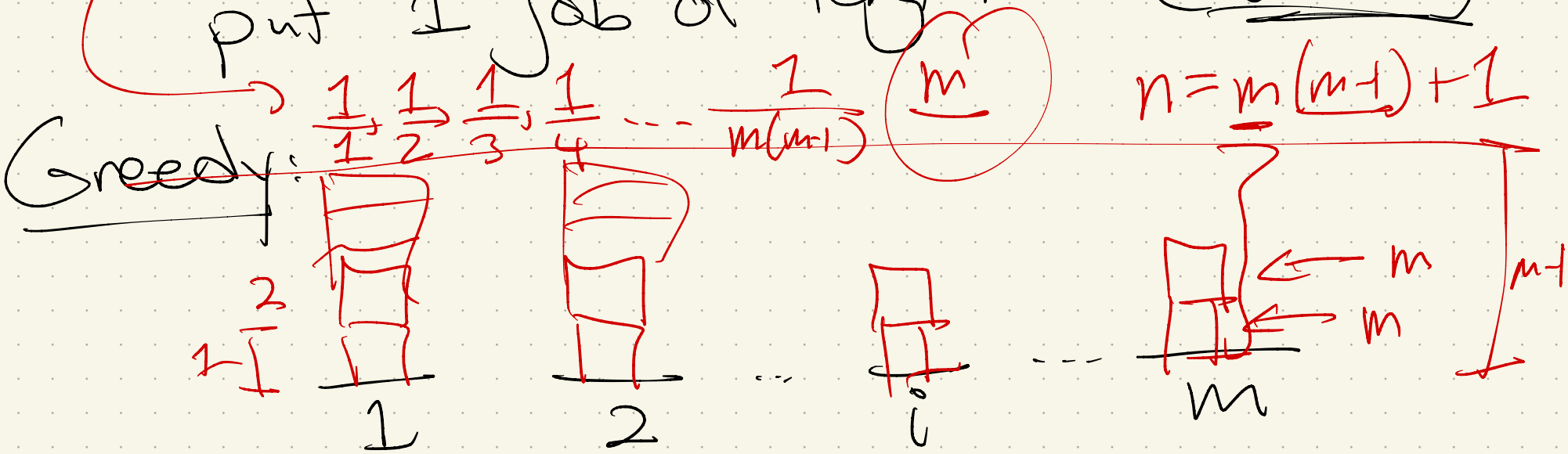
Use ②: $OPT \geq avg \geq$ machine i's load

Is this tight? adversarial!

No (but close): think lower bound

Consider $m(m-1)$ jobs of length 1

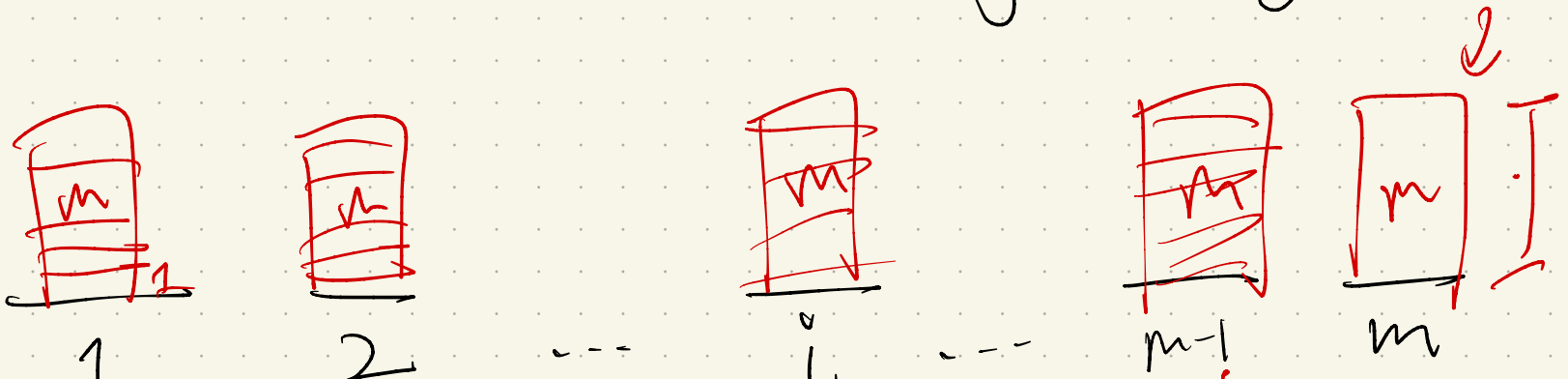
put 1 job of length m (at end)



makespan: $m-1+m = 2m-1$

greedy

What would opt do? $m(m-1)$ jobs length 1
+ 1 job length m



OPT = m

Ratio: $\frac{\text{Greedy}}{OPT} = \frac{2m-1}{m} < 2$

as $m \to \infty$, $\to 2$

Aside: Streaming

A model where inputs arrive one at a time.

Very common! examples:

- processor + jobs
- internet routing
- ⋮

Input is not known in advance

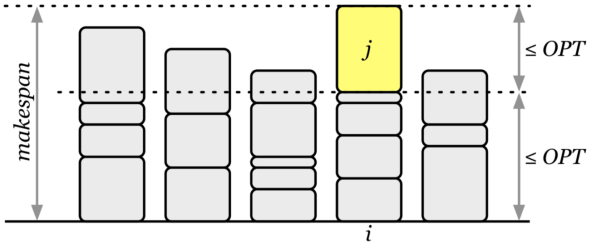
Large enough to not be able to store everything

Improvement : offline alg.

If we know jobs ahead of time, can improve → sort largest → smallest.

Then: • if #jobs ≤ # machines:

↳ every job goes to new machine



Proof that GREEDYLOADBALANCE is a 2-approximation algorithm

• if #jobs ≥ m+1:

Pigeonhole princple ⇒ some machine must run 2 jobs (or more)

↳ k & l Job

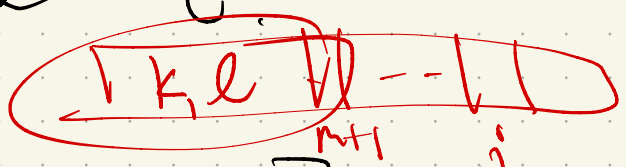
So for some k & l ≤ #jobs:

OPT ≥ T[k] + T[l] ⇒ either T[k] ≤ OPT/2 or T[l] ≤ OPT/2

either $T[k] \geq \frac{OPT}{2}$

or $T[\ell] \geq \frac{OPT}{2}$

Then: Let j be last job on longest span machine i.



$$T[j] \le T[m+1] \le T[\max\{k,l\}]$$

$$= \min\{T[k], T[l]\}$$

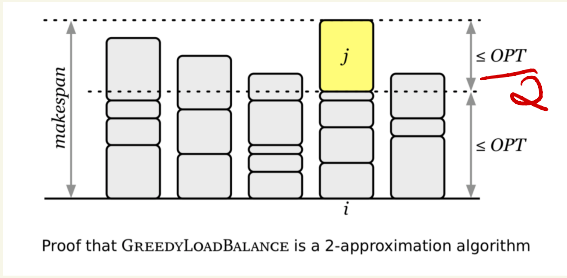
(since we sorted big → small)

Since $OPT \le T[k] + T[l]$

$$\Rightarrow \min\{T[k], T[l]\} \le \frac{OPT}{2}$$

So greedy $\le$ OPT + $\frac{OPT}{2}$

avg ↑ last job



Proof that GREEDYLOADBALANCE is a 2-approximation algorithm

Question: is 3/2 tight?

Also no! [Graham 1969] showed 4/3 bound. (not cover)

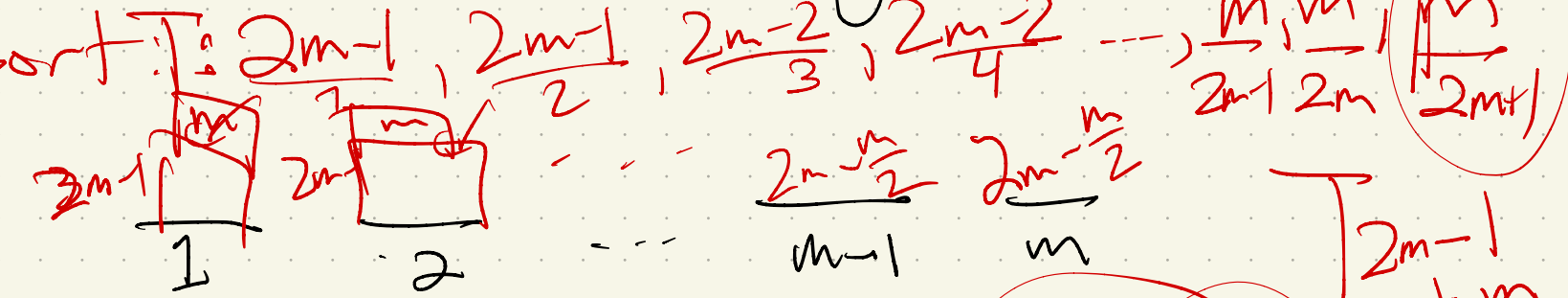
This one is close to tight:

m machines, $n = 2m+1$ jobs:

2 each of length $m, m+1, \ldots, 2m-1$

plus one extra of length m

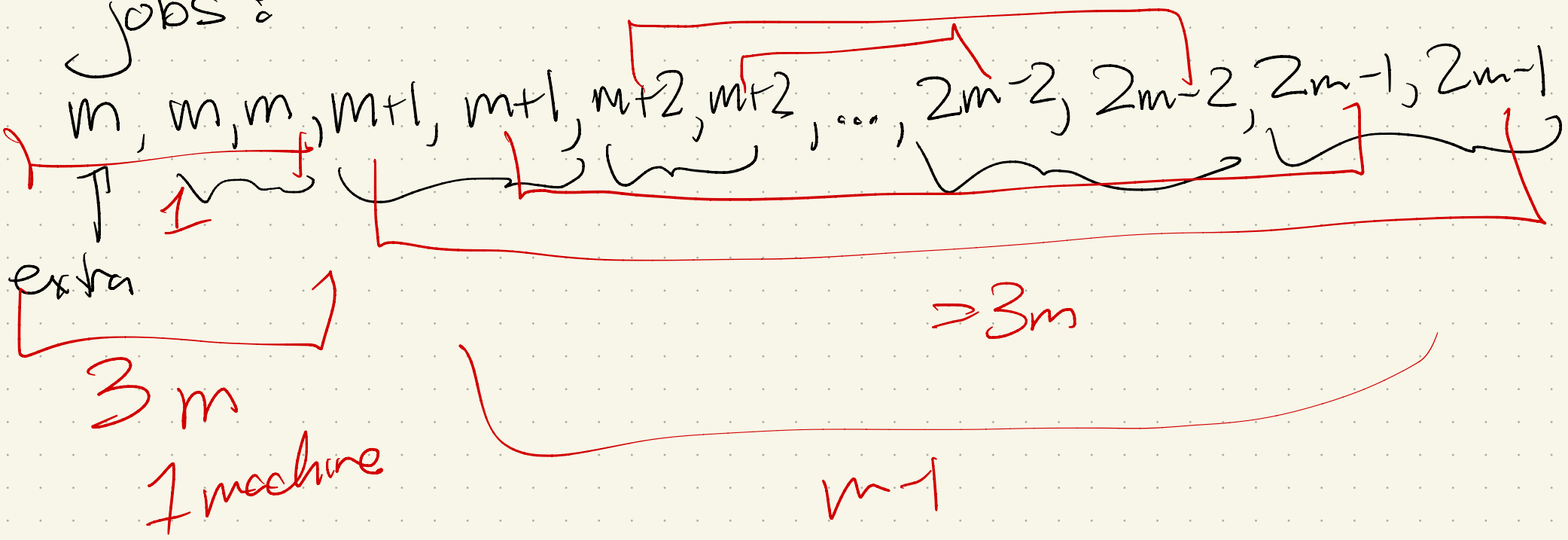
greedy:



makespan: $2m-1+m+m = 4m-1$

Opt: Can get 3m:

Jobs:



So ratio: $\frac{4m-1}{3m} \approx 4/3$

Clustering

Input: $X \subseteq \mathbb{R}^d$, $X = \{x_1, \dots, x_n\}$ + distance

$$d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

Output: A set of clusters $X_1, \dots, X_k$, each associated with a point $c_i \in \mathbb{R}^d$: $\Phi : X \to \{c_1, \dots, c_k\}$

$\Phi(x) = c_i$

Cost of cluster:

k-means: $\min \sum_{x \in X} d(x, \Phi(x))^2$

k-center: $\min \{ \max_x d(x, \Phi(x)) \}$

k-median: $\min \sum_{x \in X} d(x, \Phi(x))$

K-center

Bad news: NP-Hard

Even to approximate within a factor of $c < 2$

How?

Show a better than 2-approx would let us solve some NP-Hard problem.

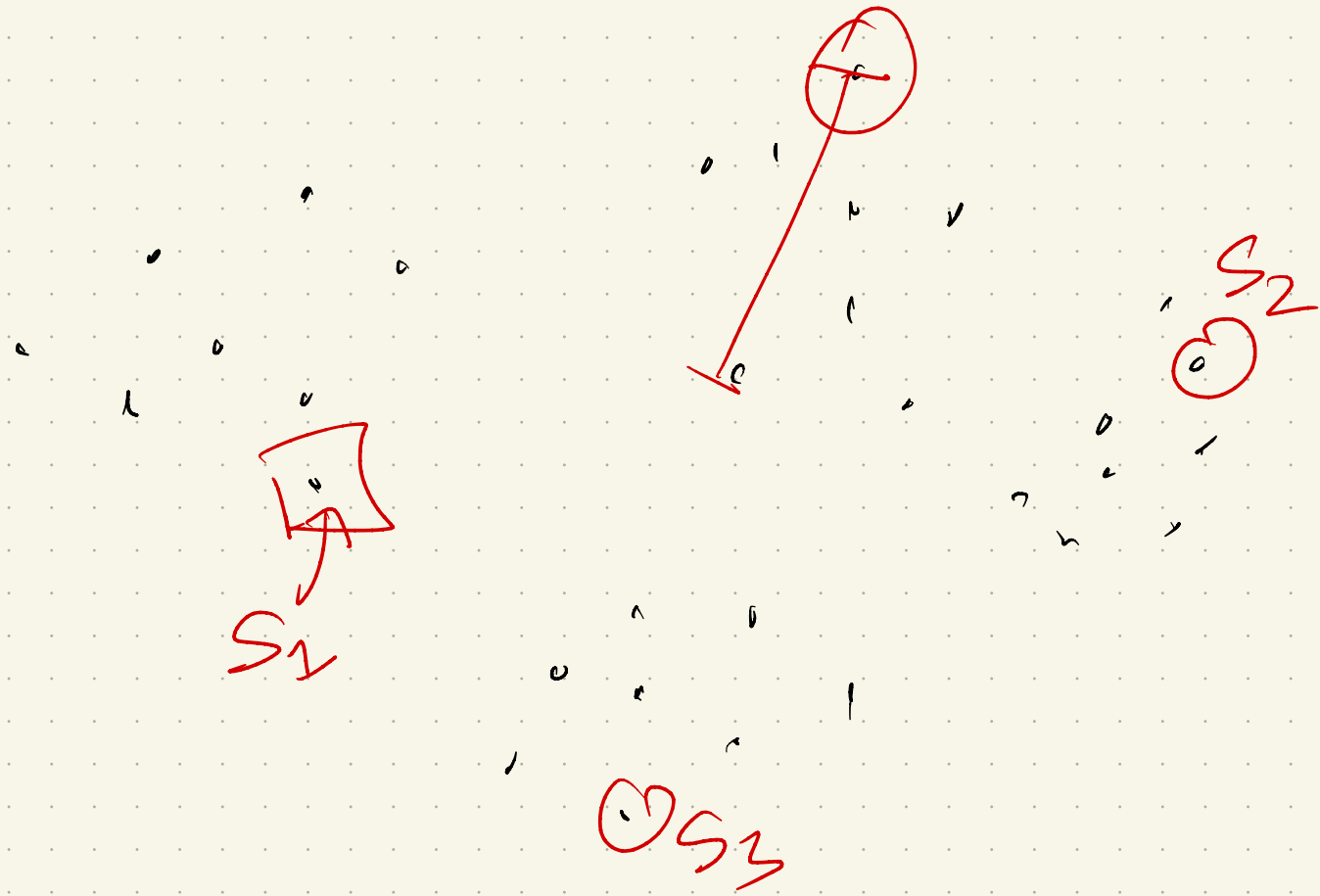
K-center: Gonzalez's algorithm

Be greedy!

- Pick random point & make it first center
- while we don't have k points
  choose furthest point away

Algorithm 8.2.1 Gonzalez Algorithm(X, k)

Choose $s_1 \in X$ arbitrarily. Initialize $S = \{s_1\}$.
for $i = 2$ to k **do**
  Set $s_i = \arg\max_{x \in X} \mathbf{d}(x, \phi_S(x))$.
  Include $S = S \cup \{s_i\}$.



K=3

K=4

Approximation Ratio : 2-approx

Consider end of algorithm.

Suppose we were to pick one more s_{k+1}

↳ some distance R away from closest center

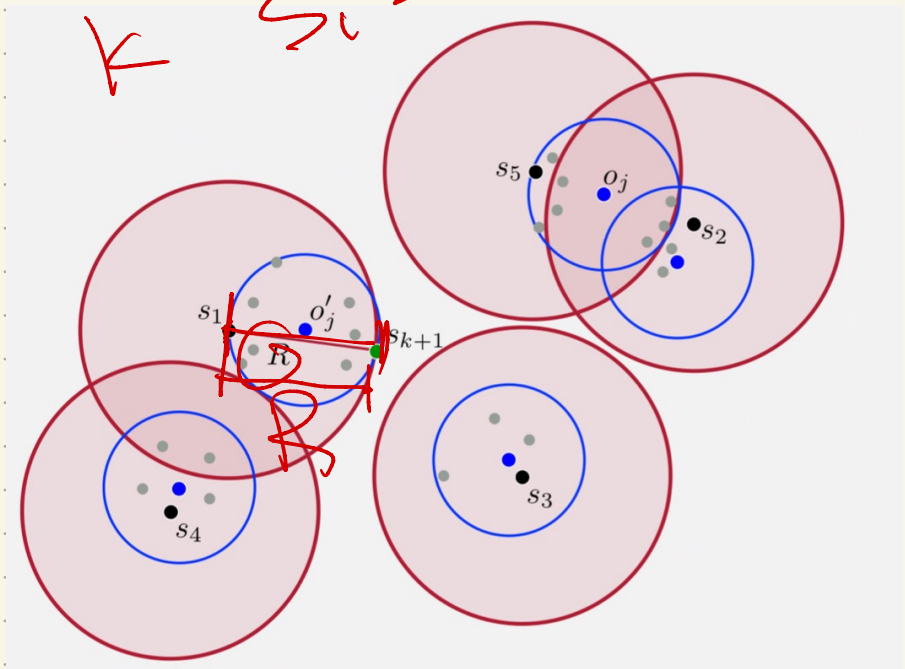
So, $d(s_i, s_j) \geq R$ (1)

for s_i's

Why?

if not,

I would have chosen s_{k+1} in my loop



Now, proof by contradiction:

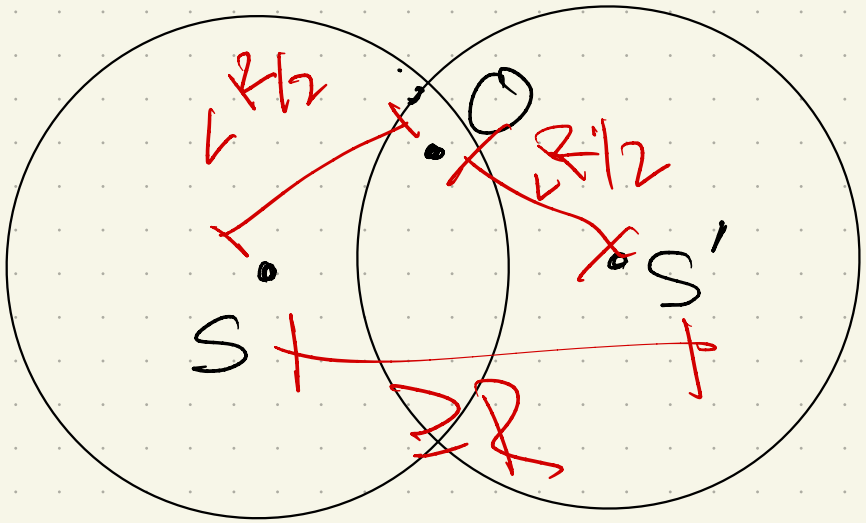
Let $o_1, \dots, o_k$ be optimal centers, &

suppose their cost is $< R/2$

↳ better than 2-approx

Claim: every site o must be closest to distinct s_i.

Why? if not: in two R-balls

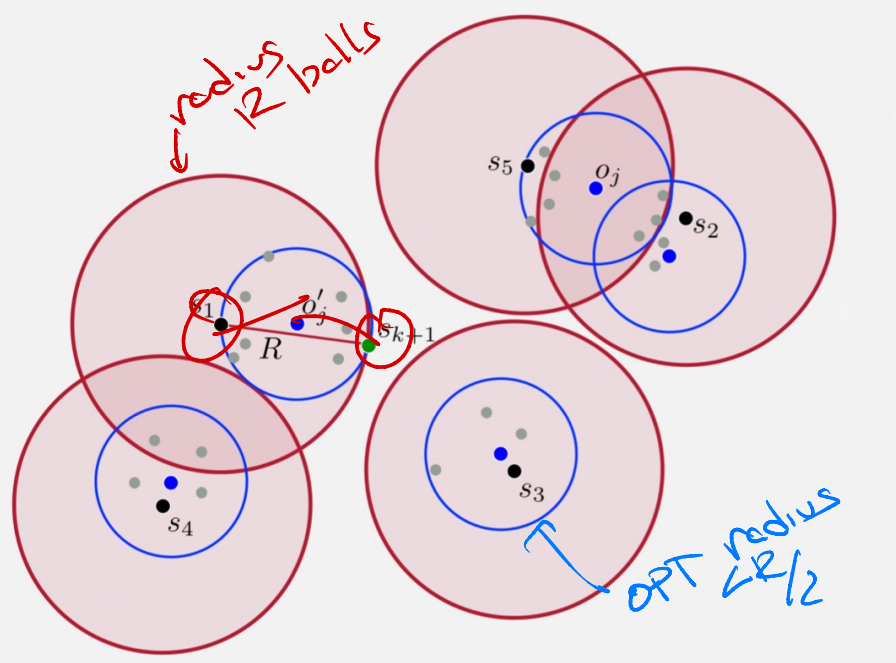


know: Δ-ineq

$$d(s,o) + d(o,s') \geq d(s,s')$$

Ok, so each optimal site o_j is in a different greedy site s_j's region!

Consider s_{k+1} again:
has $d(s_{k+1}, s_1) = R$.



So, no $o_j \in OPT$ can cover both.

But all other s_i's are $\geq R$ away from s_{k+1}

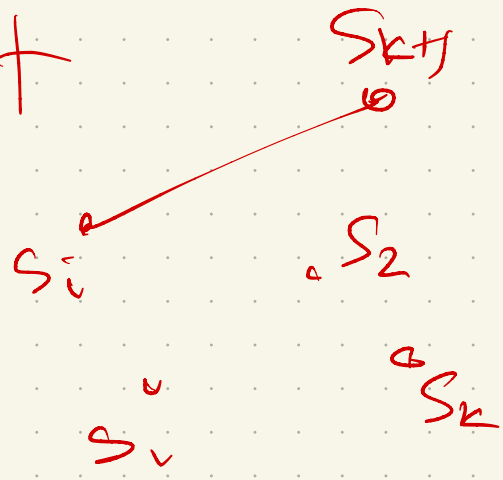
$\Rightarrow$ $k+1$ points, no o_i can cover 2
$\hookrightarrow$ need $k+1$ of the o's. $\lightning$

$\Rightarrow$ OPT must use $\geq R/2$ balls.

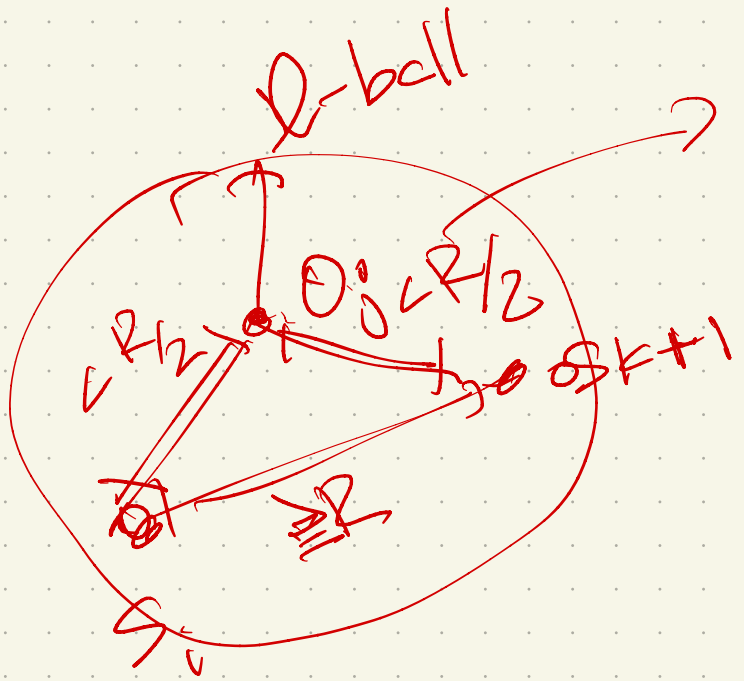
cost: $\max_{\text{point}}$ { dist to its center }

$\hookrightarrow R$ = greedy cost

$\hookrightarrow$ OPT $\geq R/2$



$\Rightarrow$ 2-approx



→ in an opt $< \frac{R}{2}$ solution

$OPT = \ell < \frac{R}{2}$