Chapter 7: Social Choice and Voting

Algorithmic Game Theory

Social choice · Plurality and majority · Borda and approval · Instant-runoff voting · Condorcet and Copeland · Scoring rules · Condorcet cycles · Kemeny · Strategic voting · Arrow and Gibbard–Satterthwaite · May's theorem · Single-peaked preferences · Computational social choice

Sercan KÜLCÜ · Giresun University · Computer Engineering

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/agt/slides/Chapter_07_Social_Choice_and_Voting.html

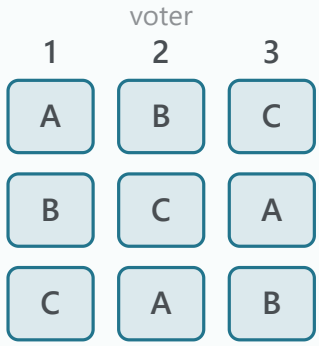
Warm-up: The Condorcet Paradox

The rules

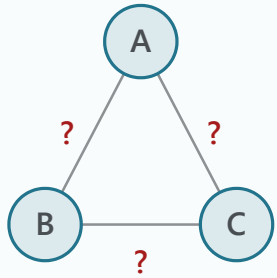
Each voter ranks the candidates. Every pair is decided by pairwise majority, and a candidate who beats every rival is the Condorcet winner.

Think about it

- If every voter's ranking is consistent, must the group's preference be consistent too?
- If you set the order of the votes, could you make any candidate you like win?



higher candidate is preferred



by pairwise majority who beats whom?

Learning Outcomes

Counting

Apply majority, plurality, Borda, approval, instant-runoff and Condorcet step by step to one preference profile.

Paradoxes

Show a Condorcet cycle, the power of agenda order and IRV's failure of monotonicity; find a Kemeny ranking.

Strategic voting

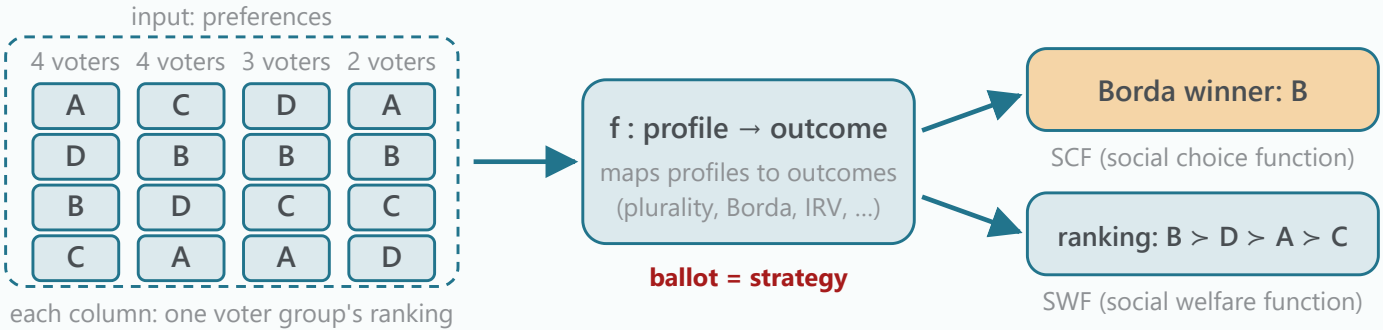
Show tactical voting under plurality and "burying" under Borda with concrete numbers.

Theorems

Explain what Arrow, Gibbard–Satterthwaite and May say; recognize single-peaked preferences as a way out.

What Is Social Choice?

- Social choice studies rules that turn individual preferences into a **collective decision** or a social ranking.
- The goal is not one universal definition of fairness, but to understand **which criteria a rule satisfies** and what trade-offs it makes.
- Applications: elections, committee and club decisions, choosing a class event, sharing common resources, online polls.



Mechanism design without money: no payments, only rankings.

A Sample Preference Profile

rank	4 voters	4 voters	3 voters	2 vo
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

A preference profile; every rule in this chapter is run on it.

13 voters, 4 candidates

Voters with the same ranking are grouped into one column.

Reading it

Column 1: 4 voters rank A ≻ D ≻ B ≻ C.

Majority threshold

More than half of 13 votes: **7 votes**.

Applied to the same profile, the rules give **four different winners**.

VOTING RULES

Basic Voting Rules

Rule	What does a voter report?	Winner
Majority	One candidate	Whoever gets **more than half** of the votes (otherwise no winner)
Plurality	One candidate	Most first-place votes; no majority needed
Borda count	Full ranking	With m candidates, positions score $m-1$, $m-2$, …, 0; highest total wins
Approval	Set of approved candidates	Most approvals
Instant-runoff (IRV)	Full ranking	Last place is eliminated and its votes transfer; repeat until a majority
Condorcet	Full ranking	The candidate who **beats every rival** head to head (may not exist)

Note: "most votes wins" is plurality; majority requires more than 50% of the votes.

Majority and Plurality Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	votes
A	?
B	?
C	?
D	?

Under majority and plurality each voter reports only a **first choice**; the rest of the ranking is ignored.

Majority and Plurality Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	first choice
A	**6**
B	0
C	4
D	3

Count first places: A = 4 + 2 = 6, C = 4, D = 3, B = 0.

Majority and Plurality Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	first choice
A	**6**
B	0
C	4
D	3

Majority: $6 < 7 \Rightarrow$ **no winner**. Plurality: **A** has the most votes and wins (with just 6/13).

Majority and Plurality Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	~~D~~	**A**
2.	~~D~~	~~B~~	~~B~~	~~B~~
3.	~~B~~	~~D~~	**C**	C
4.	C	A	A	~~D~~

candidate	runoff
A	6
B	—
C	**7**
D	—

Two-round system (e.g. a presidential election): the top two, A and C, go to a runoff. A = 6, C = 7 ⇒ **C** wins.

Borda Count Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	total
A	0
B	0
C	0
D	0

Borda with 4 candidates: 1st place gets **3**, 2nd **2**, 3rd **1**, 4th **0** points. A group adds voters × points.

Borda Count Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	G1	total
A	12	12
B	4	4
C	0	0
D	8	8

G1–G4: group contributions

Group 1 (4 voters, A ≻ D ≻ B ≻ C): A: 4×3 = 12, D: 4×2 = 8, B: 4×1 = 4, C: 4×0 = 0.

Borda Count Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	**C**	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	G1	G2	total
A	12	0	12
B	4	8	12
C	0	12	12
D	8	4	12

G1–G4: group contributions

Group 2 (4 voters, C ≻ B ≻ D ≻ A): C: 4×3 = 12, B: 4×2 = 8, D: 4×1 = 4, A: 4×0 = 0.

Borda Count Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	**D**	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	G1	G2	G3	total
A	12	0	0	12
B	4	8	6	18
C	0	12	3	15
D	8	4	9	21

G1–G4: group contributions

Group 3 (3 voters, D ≻ B ≻ C ≻ A): D: 3×3 = 9, B: 3×2 = 6, C: 3×1 = 3, A: 3×0 = 0.

Borda Count Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	**A**
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	G1	G2	G3	G4	total
A	12	0	0	6	18
B	4	8	6	4	22
C	0	12	3	2	17
D	8	4	9	0	21

G1–G4: group contributions

Group 4 (2 voters, A ≻ B ≻ C ≻ D): A: 2×3 = 6, B: 2×2 = 4, C: 2×1 = 2, D: 2×0 = 0.

Borda Count Step by Step

rank	4 voters	4 voters	3 voters	2 vo	candidate	G1	G2	G3	G4	total
1.	A	C	D	A	**A**	12	0	0	6	18
2.	D	B	B	B	**B**	**4**	**8**	**6**	**4**	**22**
3.	B	D	C	C	**C**	0	12	3	2	17
4.	C	A	A	D	**D**	8	4	9	0	21

Totals: B = 22, D = 21, A = 18, C = 17 ⇒ Borda winner **B**, yet nobody ranked B first!

Approval Voting

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	approval
A	?
B	?
C	?
D	?

In approval voting each voter marks **every** candidate they accept. Assume each voter approves their top two.

Approval Voting

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	**D**	**B**	**B**	**B**
3.	B	D	C	C
4.	C	A	A	D

candidate	approval
A	6
B	**9**
C	4
D	7

Approvals: A = 4 + 2 = 6, B = 4 + 3 + 2 = 9, C = 4, D = 4 + 3 = 7 ⇒ approval winner **B** (if everyone approves their top two, approval becomes 2-approval).

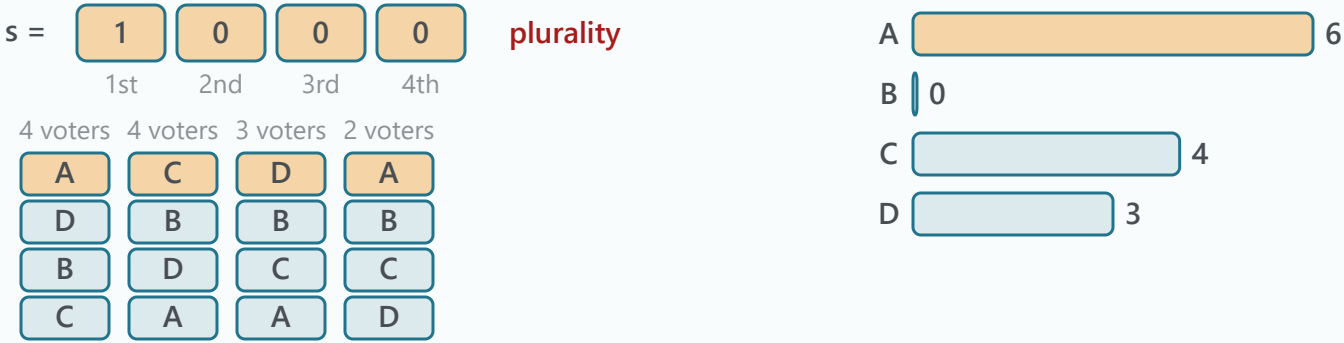
Approval Voting

rank	4 voters	4 voters	3 voters	2 voters	candidate	approval (1st only)
1.	**A**	**C**	**D**	**A**	**A**	**6**
2.	D	B	B	B	**B**	0
3.	B	D	C	C	**C**	4
4.	C	A	A	D	**D**	3

The result depends on the threshold: if everyone approved only their first choice, approval = plurality and A would win.

Positional Scoring Rules

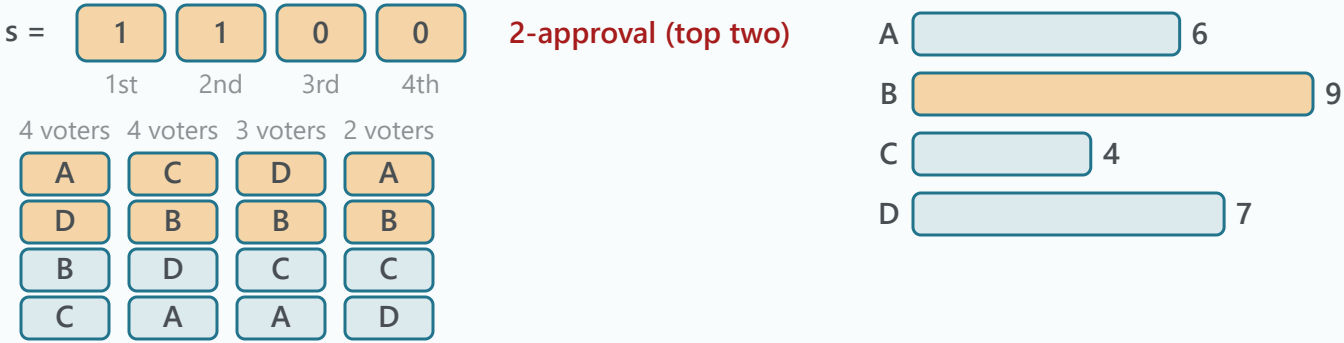
A positional scoring rule fixes a vector $s = (s_1 \geq s_2 \geq \ldots \geq s_m)$ for m candidates; a candidate in position k gets s_k points. Plurality, k-approval, Borda and veto all belong to this family.



s = (1, 0, 0, 0) (plurality): A = 6, C = 4, D = 3, B = 0 ⇒ winner **A**. Orange cells are the scoring positions.

Positional Scoring Rules

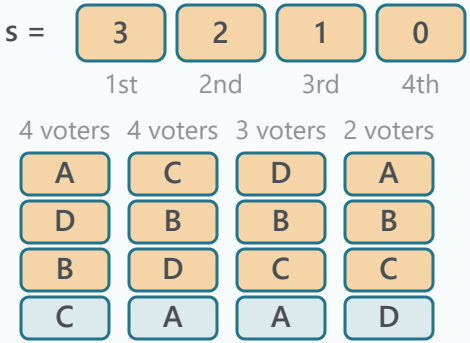
A positional scoring rule fixes a vector $s = (s_1 \geq s_2 \geq \ldots \geq s_m)$ for m candidates; a candidate in position k gets s_k points. Plurality, k-approval, Borda and veto all belong to this family.



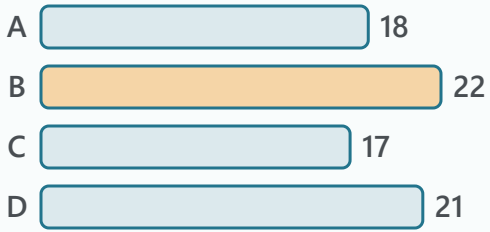
s = (1, 1, 0, 0) (2-approval): B = 9, D = 7, A = 6, C = 4 ⇒ winner **B**.

Positional Scoring Rules

A positional scoring rule fixes a vector $s = (s_1 \geq s_2 \geq \ldots \geq s_m)$ for m candidates; a candidate in position k gets s_k points. Plurality, k-approval, Borda and veto all belong to this family.



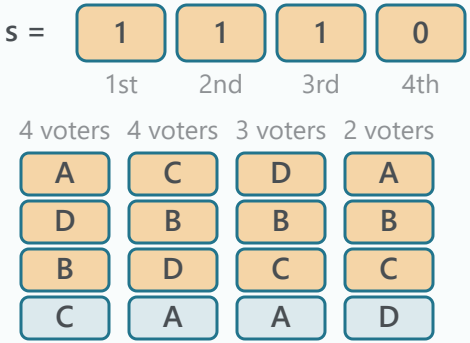
Borda



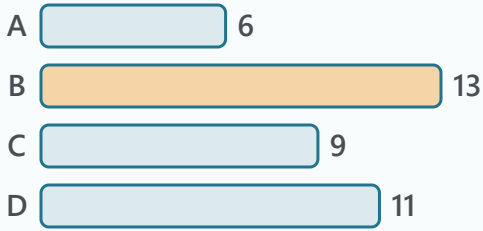
$s = (3, 2, 1, 0)$ (Borda): $B = 22$, $D = 21$, $A = 18$, $C = 17 \Rightarrow$ winner **B**.

Positional Scoring Rules

A positional scoring rule fixes a vector $s = (s_1 \geq s_2 \geq \ldots \geq s_m)$ for m candidates; a candidate in position k gets s_k points. Plurality, k-approval, Borda and veto all belong to this family.



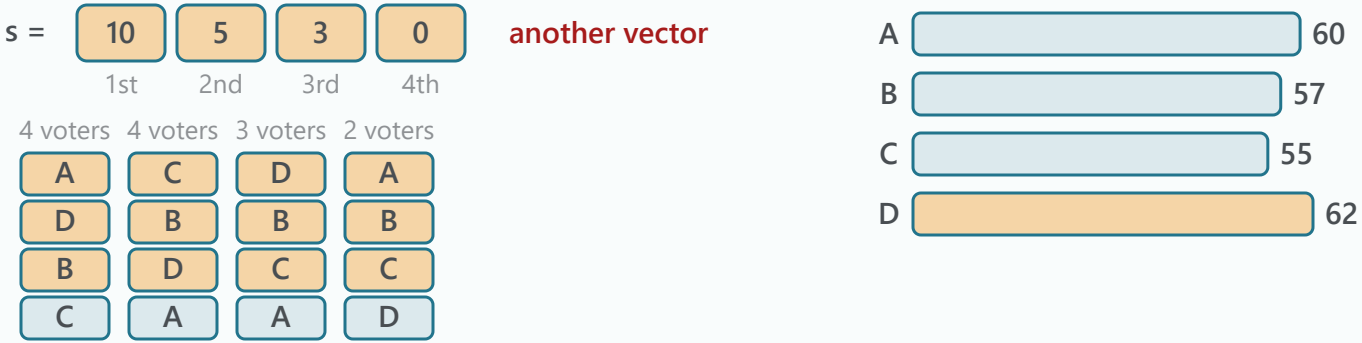
veto (0 to last)



s = (1, 1, 1, 0) (veto): B = 13, D = 11, C = 9, A = 6 ⇒ winner **B**.

Positional Scoring Rules

A positional scoring rule fixes a vector $s = (s_1 \geq s_2 \geq \ldots \geq s_m)$ for m candidates; a candidate in position k gets s_k points. Plurality, k-approval, Borda and veto all belong to this family.



s = (10, 5, 3, 0) (another vector): D = 62, A = 60, B = 57, C = 55 ⇒ winner **D**. This vector picks the Condorcet winner D.

Instant-Runoff Voting (IRV) Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	round 1
A	?
B	?
C	?
D	?

Instant-runoff (ranked-choice) voting: each round counts every voter's top candidate **still in the race**; with no majority, the weakest is eliminated.

Instant-Runoff Voting (IRV) Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	round 1
A	6
B	0
C	4
D	3

Round 1: A = 6, B = 0, C = 4, D = 3. No majority (7 needed).

Instant-Runoff Voting (IRV) Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	D	~~B~~	~~B~~	~~B~~
3.	~~B~~	D	C	C
4.	C	A	A	D

candidate	round 1
A	6
B	0
C	4
D	3

B has the fewest votes (0) and is eliminated. Nobody ranked B first, so no votes transfer.

Instant-Runoff Voting (IRV) Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	D	~~B~~	~~B~~	~~B~~
3.	~~B~~	D	C	C
4.	C	A	A	D

candidate	round 2
A	6
B	—
C	4
D	3

Round 2: A = 6, C = 4, D = 3. No majority (7 needed).

Instant-Runoff Voting (IRV) Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	~~D~~	**A**
2.	~~D~~	~~B~~	~~B~~	~~B~~
3.	~~B~~	~~D~~	**C**	C
4.	C	A	A	~~D~~

candidate	round 2
A	6
B	—
C	4
D	3

D has the fewest (3) and is eliminated. Its votes go to the next remaining choice: 3 votes D → C.

Instant-Runoff Voting (IRV) Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	~~D~~	**A**
2.	~~D~~	~~B~~	~~B~~	~~B~~
3.	~~B~~	~~D~~	**C**	C
4.	C	A	A	~~D~~

candidate	round 3
A	6
B	—
C	**7**
D	—

Round 3: A = 6, C = 7. 7 ≥ 7 ⇒ **C** wins with a majority.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	A	C	D	A	**A**	—				0
2.	D	B	B	B	**B**		—			0
3.	B	D	C	C	**C**			—		0
4.	C	A	A	D	**D**				—	0

Condorcet: for each pair, "how many voters prefer x to y?" Cell (x, y) = number of voters preferring x to y. 4 candidates ⇒ 6 comparisons.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	**A**	C	D	**A**	**A**	—	6			0
2.	D	**B**	**B**	**B**	**B**	**7**	—			1
3.	**B**	D	C	C	**C**			—		0
4.	C	**A**	**A**	D	**D**				—	0

A – B: 6 prefer A, 7 prefer B ⇒ **B** wins.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	**A**	**C**	D	**A**	**A**	—	6	6		0
2.	D	B	B	B	**B**	7	—			1
3.	B	D	**C**	**C**	**C**	**7**		—		1
4.	**C**	**A**	**A**	D	**D**				—	0

A – C: 6 prefer A, 7 prefer C ⇒ **C** wins.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	**A**	C	**D**	**A**	**A**	—	6	6	6	0
2.	**D**	B	B	B	**B**	7	—			1
3.	B	**D**	C	C	**C**	7		—		1
4.	C	**A**	**A**	**D**	**D**	**7**			—	1

A – D: 6 prefer A, 7 prefer D ⇒ **D** wins.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	A	**C**	D	A	**A**	—	6	6	6	0
2.	D	**B**	**B**	**B**	**B**	7	—	**9**		2
3.	**B**	D	**C**	**C**	**C**	7	4	—		1
4.	**C**	A	A	D	**D**	7			—	1

B – C: 9 prefer B, 4 prefer C ⇒ **B** wins.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	A	C	**D**	A	**A**	—	6	6	6	0
2.	**D**	**B**	**B**	**B**	**B**	7	—	9	6	2
3.	**B**	**D**	C	C	**C**	7	4	—		1
4.	C	A	A	**D**	**D**	7	**7**		—	2

B – D: 6 prefer B, 7 prefer D ⇒ **D** wins.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	A	**C**	**D**	A	**A**	—	6	6	6	0
2.	**D**	B	B	B	**B**	7	—	9	6	2
3.	B	**D**	**C**	**C**	**C**	7	4	—	6	1
4.	**C**	A	A	**D**	**D**	7	7	**7**	—	3

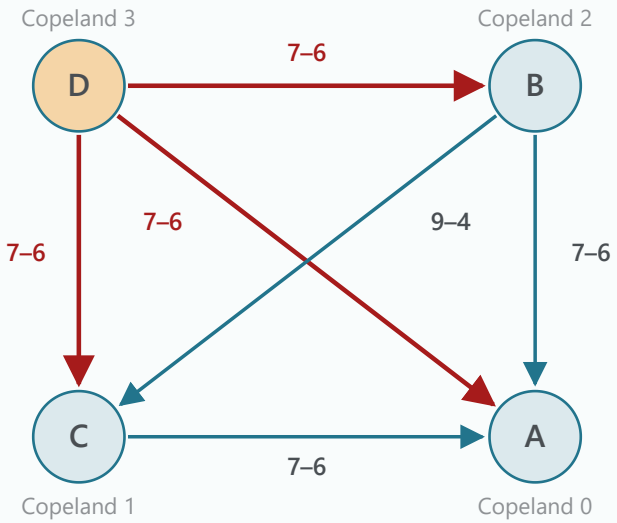
C – D: 6 prefer C, 7 prefer D ⇒ **D** wins.

Condorcet: Pairwise Comparisons Step by Step

rank	4 voters	4 voters	3 voters	2 voters	x ╲ y	A	B	C	D	wins
1.	A	C	D	A	**A**	—	6	6	6	0
2.	D	B	B	B	**B**	7	—	9	6	2
3.	B	D	C	C	**C**	7	4	—	6	1
4.	C	A	A	D	**D**	**7**	**7**	**7**	**—**	**3**

D beats all three rivals 7–6: D is the Condorcet winner. A loses to everyone (the Condorcet loser), yet A won plurality!

The Pairwise Majority Graph and Copeland



- Each candidate is a node; an arrow **x** → **y** means a majority prefers x to y (label: vote counts).
- The graph is a **tournament**: exactly one arrow between every pair (an odd number of voters rules out ties).
- **Condorcet winner** = the node with no incoming arrows (D). Condorcet loser = the node with only incoming arrows (A).
- **Copeland** rule: most outgoing arrows (wins) wins ⇒ D (3) ≻ B (2) ≻ C (1) ≻ A (0).

If every candidate has an incoming arrow, there is no Condorcet winner; a **cycle** through all the candidates causes exactly this.

One Profile, Four Different Winners

Rule	Count	Winner
Majority	A 6 < 7	none
Plurality	A 6, C 4, D 3, B 0	**A**
Two-round system	C 7 – A 6	**C**
Instant-runoff	C 7 – A 6 (round 3)	**C**
Borda	B 22, D 21, A 18, C 17	**B**
2-approval (top two)	B 9, D 7, A 6, C 4	**B**
Condorcet	D 3 wins	**D**
Kemeny	D ≻ B ≻ C ≻ A (score 44)	**D**

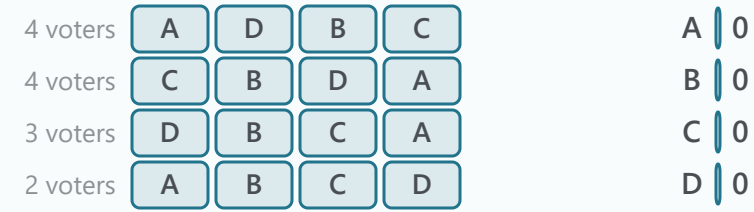
Lesson 1

"The will of the people" depends on the rule: the rule shapes the outcome as much as the voters do.

Lesson 2

Plurality can elect the Condorcet *loser* (A); IRV can eliminate the Condorcet *winner* (D).

Code Trace: `borda(ballots)`



ballots: row = voter group; 1st place 3 points ... 4th 0

ballots = 4 groups (rows). The `score` dict starts empty and accumulates points per candidate.

Step 1 / 11

```
▶ def borda(ballots):
▶   score = {}
3   for count, order in ballots:
4     m = len(order)
5     for k, cand in enumerate(order):
6       p = count * (m - 1 - k)
7       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`score = {}`

Code Trace: `borda(ballots)`

4 voters	A	D	B	C	A | 0
4 voters	C	B	D	A	B | 0
3 voters	D	B	C	A	C | 0
2 voters	A	B	C	D	D | 0

ballots: row = voter group; 1st place 3 points ... 4th 0

Group 1: 4 voters, ranking ADBC. m = 4 candidates: position k earns m − 1 − k.

Step 2 / 11

```
1 def borda(ballots):
2   score = {}
▶   for count, order in ballots:
▶     m = len(order)
5     for k, cand in enumerate(order):
6       p = count * (m - 1 - k)
7       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`count = 4` `order = "ADBC"` `m = 4` `score = {}`

Code Trace: `borda(ballots)`

4 voters	A	D	B	C
4 voters	C	B	D	A
3 voters	D	B	C	A
2 voters	A	B	C	D



ballots: row = voter group; 1st place 3 points … 4th 0

k = 0: A is 1st ⇒ p = 4 × (3 − 0) = **12**; A total 12.

Step 3 / 11

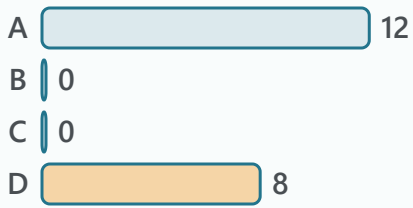
```
1 def borda(ballots):
2   score = {}
3   for count, order in ballots:
4     m = len(order)
▶     for k, cand in enumerate(order):
▶       p = count * (m - 1 - k)
▶       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`count = 4` `order = "ADBC"` `k = 0` `cand = A` `p = 12` `score = {A: 12}`

Code Trace: `borda(ballots)`

4 voters	A	D	B	C
4 voters	C	B	D	A
3 voters	D	B	C	A
2 voters	A	B	C	D



ballots: row = voter group; 1st place 3 points … 4th 0

k = 1: D is 2nd ⇒ p = 4 × (3 − 1) = **8**; D total 8.

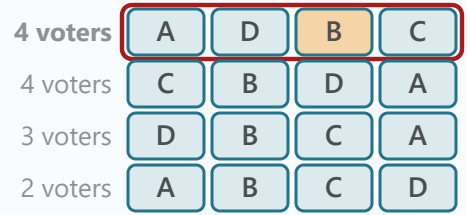
Step 4 / 11

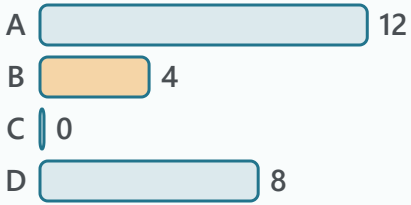
```
1 def borda(ballots):
2   score = {}
3   for count, order in ballots:
4     m = len(order)
      for k, cand in enumerate(order):
        p = count * (m - 1 - k)
        score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`count = 4` `order = "ADBC"` `k = 1` `cand = D` `p = 8` `score = {A: 12, D: 8}`

Code Trace: `borda(ballots)`





ballots: row = voter group; 1st place 3 points ... 4th 0

k = 2: B is 3rd ⇒ p = 4 × (3 − 2) = **4**; B total 4.


```
1 def borda(ballots):
2   score = {}
3   for count, order in ballots:
4     m = len(order)
▶     for k, cand in enumerate(order):
▶       p = count * (m - 1 - k)
▶       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

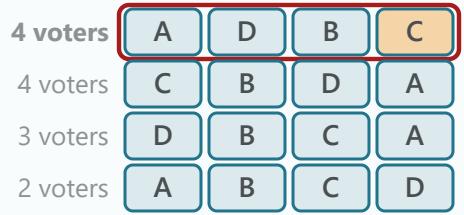
VARIABLES

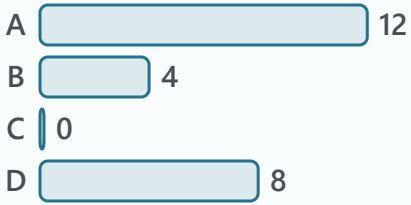
`count = 4` `order = "ADBC"`

`k = 2` `cand = B` `p = 4`

`score = {A: 12, D: 8, B: 4}`

Code Trace: `borda(ballots)`





ballots: row = voter group; 1st place 3 points … 4th 0

k = 3: C is 4th ⇒ p = 4 × (3 − 3) = **0**; C total 0.


```
1 def borda(ballots):
2   score = {}
3   for count, order in ballots:
4     m = len(order)
▶     for k, cand in enumerate(order):
▶       p = count * (m - 1 - k)
▶       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

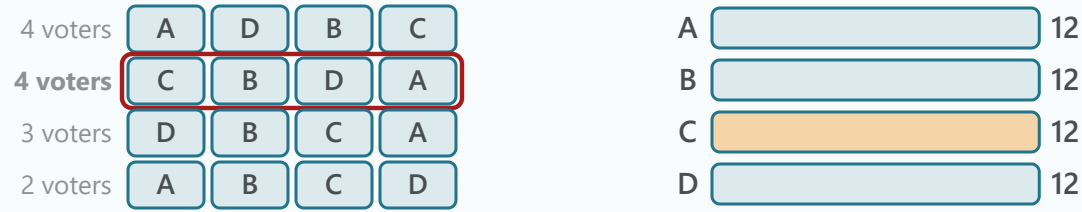
VARIABLES

`count = 4` `order = "ADBC"`

`k = 3` `cand = C` `p = 0`

`score = {A: 12, D: 8, B: 4, C: 0}`

Code Trace: `borda(ballots)`



ballots: row = voter group; 1st place 3 points … 4th 0

Group 2 (4 × CBDA): the inner loop runs 4 times; C +12, B +8, D +4, A +0.

Step 7 / 11

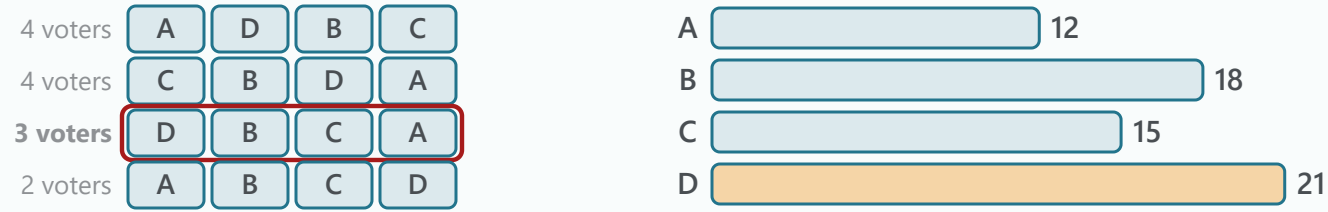
```
1 def borda(ballots):
2   score = {}
▶   for count, order in ballots:
▶     m = len(order)
▶     for k, cand in enumerate(order):
▶       p = count * (m - 1 - k)
▶       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`count = 4` `order = "CBDA"` `m = 4`

`score = {A: 12, D: 12, B: 12, C: 12}`

Code Trace: `borda(ballots)`



ballots: row = voter group; 1st place 3 points … 4th 0

Group 3 (3 × DBCA): the inner loop runs 4 times; D +9, B +6, C +3, A +0.

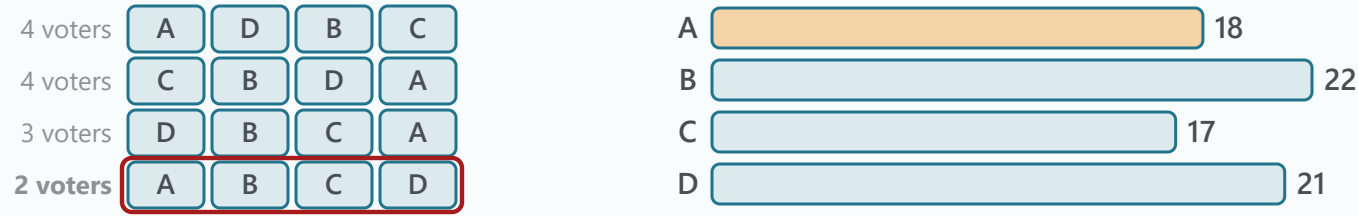

```
1 def borda(ballots):
2   score = {}
▶   for count, order in ballots:
▶     m = len(order)
▶     for k, cand in enumerate(order):
▶       p = count * (m - 1 - k)
▶       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`count = 3` `order = "DBCA"` `m = 4`

`score = {A: 12, D: 21, B: 18, C: 15}`

Code Trace: `borda(ballots)`



ballots: row = voter group; 1st place 3 points … 4th 0

Group 4 (2 × ABCD): the inner loop runs 4 times; A +6, B +4, C +2, D +0.


```
1 def borda(ballots):
2   score = {}
▶   for count, order in ballots:
▶     m = len(order)
▶     for k, cand in enumerate(order):
▶       p = count * (m - 1 - k)
▶       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
9   return winner, score
```

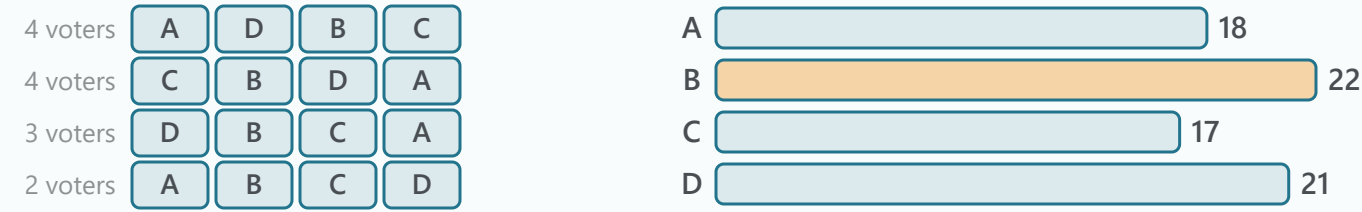
VARIABLES

`count = 2` `order = "ABCD"`

`m = 4`

`score = {A: 18, D: 21, B: 22, C: 17}`

Code Trace: `borda(ballots)`



ballots: row = voter group; 1st place 3 points … 4th 0

`max` : the top scorer is **B** (22), whom no voter ranked first. On ties `max` returns the first it sees; real rules need a tie-breaking rule.

Step 10 / 11

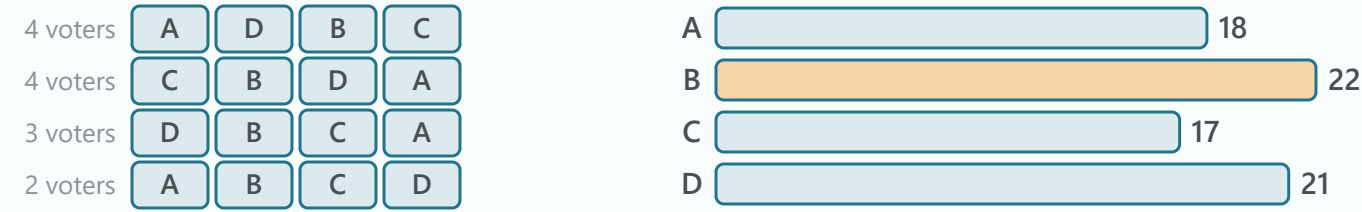
```
1 def borda(ballots):
2   score = {}
3   for count, order in ballots:
4     m = len(order)
5     for k, cand in enumerate(order):
6       p = count * (m - 1 - k)
7       score[cand] = score.get(cand, 0) + p
▶   winner = max(score, key=score.get)
9   return winner, score
```

VARIABLES

`score = {A: 18, D: 21, B: 22, C: 17}`

`winner = B`

Code Trace: `borda(ballots)`



ballots: row = voter group; 1st place 3 points … 4th 0

Complexity: n voter groups, m candidates ⇒ O(n·m). Counting is easy; the hard part is the rule's properties.

Step 11 / 11

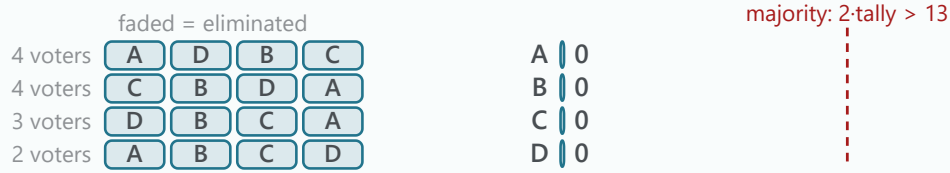
```
1 def borda(ballots):
2   score = {}
3   for count, order in ballots:
4     m = len(order)
5     for k, cand in enumerate(order):
6       p = count * (m - 1 - k)
7       score[cand] = score.get(cand, 0) + p
8   winner = max(score, key=score.get)
▶   return winner, score
```

VARIABLES

`score = {A: 18, D: 21, B: 22, C: 17}`

`winner = B`

Code Trace: `irv(ballots, cands)`



All candidates are in the race. Each round: count, check for a majority, otherwise drop the weakest.

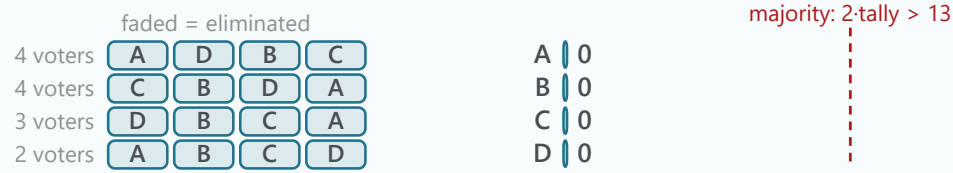
Step 1 / 15

```
▶ def irv(ballots, cands):
▶   remaining = list(cands)
3   while True:
4     tally = {a: 0 for a in remaining}
5     for count, order in ballots:
6       top = next(a for a in order if a in remaining)
7       tally[top] += count
8     leader = max(tally, key=tally.get)
9     if 2 * tally[leader] > sum(tally.values()):
10      return leader
11    loser = min(tally, key=tally.get)
12    remaining.remove(loser)
```

VARIABLES

`remaining = {A, B, C, D}`

Code Trace: `irv(ballots, cands)`



Round 1 begins: a zero counter for each remaining candidate.

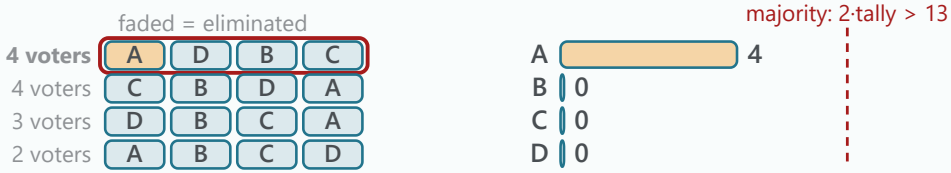

```
1  def irv(ballots, cands):
2    remaining = list(cands)
▶    while True:
▶      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

`remaining = {A, B, C, D}`

`tally = {A: 0, B: 0, C: 0, D: 0}`

Code Trace: `irv(ballots, cands)`



ADBC: top remaining candidate A ⇒ tally[A] += 4.

Step 3 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
▶      for count, order in ballots:
▶        top = next(a for a in order if a in remaining)
▶        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

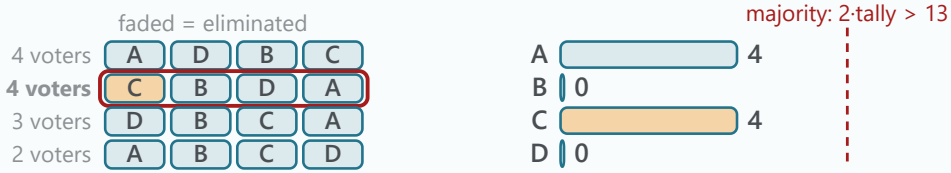
`remaining = {A, B, C, D}`

`count = 4` `order = "ADBC"`

`top = A`

`tally = {A: 4, B: 0, C: 0, D: 0}`

Code Trace: `irv(ballots, cands)`



CBDA: top remaining candidate C ⇒ tally[C] += 4.

Step 4 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
▶      for count, order in ballots:
▶        top = next(a for a in order if a in remaining)
▶        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

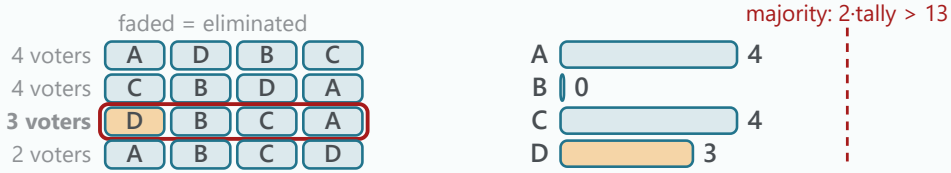
`remaining = {A, B, C, D}`

`count = 4` `order = "CBDA"`

`top = C`

`tally = {A: 4, B: 0, C: 4, D: 0}`

Code Trace: `irv(ballots, cands)`



DBCA: top remaining candidate D ⇒ tally[D] += 3.

Step 5 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
▶      for count, order in ballots:
▶        top = next(a for a in order if a in remaining)
▶        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

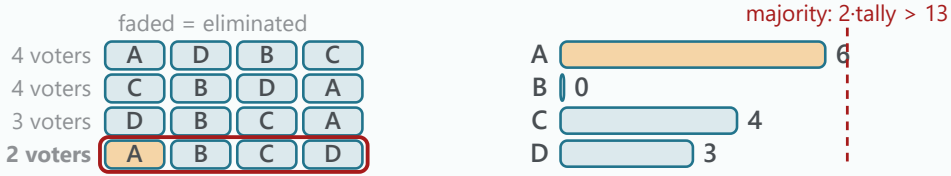
`remaining = {A, B, C, D}`

`count = 3` `order = "DBCA"`

`top = D`

`tally = {A: 4, B: 0, C: 4, D: 3}`

Code Trace: `irv(ballots, cands)`



ABCD: top remaining candidate A ⇒ tally[A] += 2.

Step 6 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
▶      for count, order in ballots:
▶        top = next(a for a in order if a in remaining)
▶        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

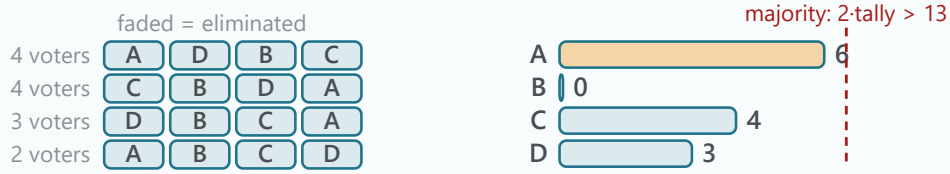
`remaining = {A, B, C, D}`

`count = 2` `order = "ABCD"`

`top = A`

`tally = {A: 6, B: 0, C: 4, D: 3}`

Code Trace: `irv(ballots, cands)`



leader = A: 2 × 6 = 12 ≤ 13 ⇒ no majority.

Step 7 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
▶      leader = max(tally, key=tally.get)
▶      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

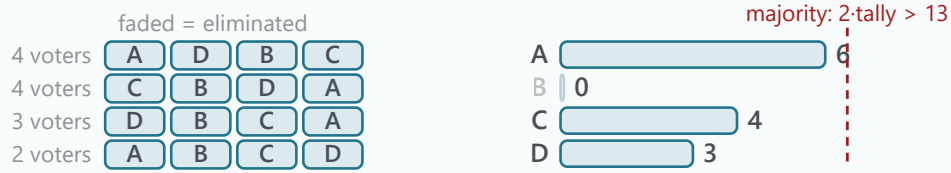
VARIABLES

`remaining = {A, B, C, D}`

`tally = {A: 6, B: 0, C: 4, D: 3}`

`leader = A`

Code Trace: `irv(ballots, cands)`



B has the fewest votes (0) and is dropped; remaining = {A, C, D}. On ties `min` and `max` return the first they see; real rules need a tie-break.

Step 8 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
▶      loser = min(tally, key=tally.get)
▶      remaining.remove(loser)
```

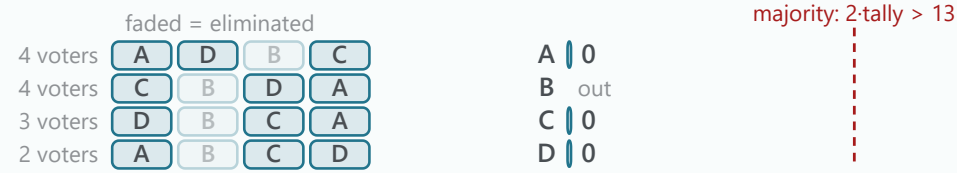
VARIABLES

`remaining = {A, C, D}`

`tally = {A: 6, B: 0, C: 4, D: 3}`

`loser = B`

Code Trace: `irv(ballots, cands)`



Round 2 begins: a zero counter for each remaining candidate.

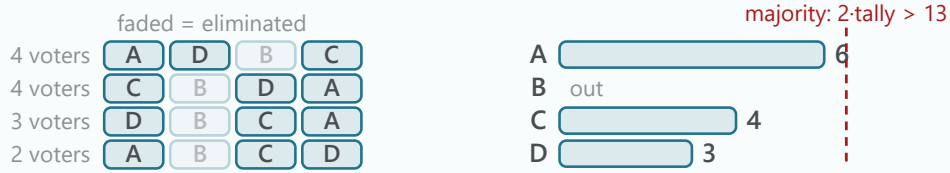

```
1  def irv(ballots, cands):
2    remaining = list(cands)
▶    while True:
▶      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

`remaining = {A, C, D}`

`tally = {A: 0, C: 0, D: 0}`

Code Trace: `irv(ballots, cands)`



Round 2 count: nobody ranked the eliminated B first, so the counts are unchanged.

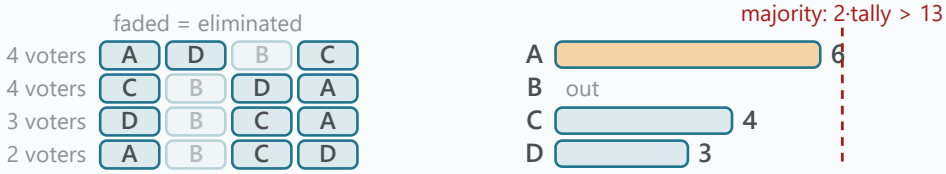

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
▶      for count, order in ballots:
▶        top = next(a for a in order if a in remaining)
▶        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

`remaining = {A, C, D}`

`tally = {A: 6, C: 4, D: 3}`

Code Trace: `irv(ballots, cands)`



leader = A: 2 × 6 = 12 ≤ 13 ⇒ no majority.

Step 11 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
▶      leader = max(tally, key=tally.get)
▶      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

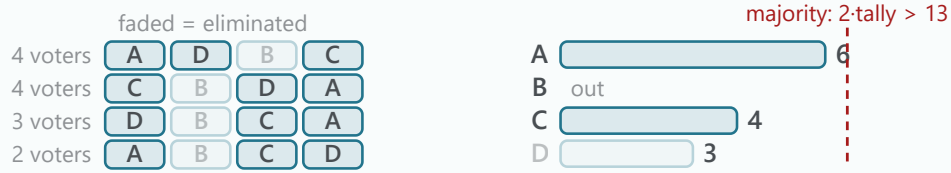
VARIABLES

`remaining = {A, C, D}`

`tally = {A: 6, C: 4, D: 3}`

`leader = A`

Code Trace: `irv(ballots, cands)`



D has the fewest (3) and is dropped; remaining = {A, C}.

Step 12 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
▶      loser = min(tally, key=tally.get)
▶      remaining.remove(loser)
```

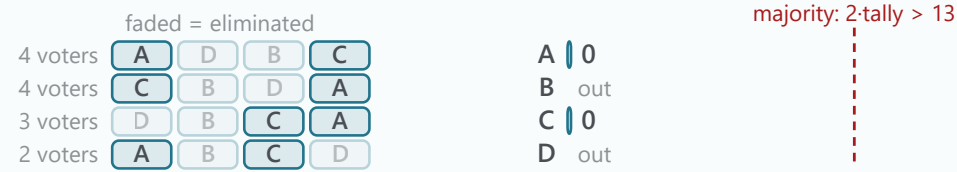
VARIABLES

`remaining = {A, C}`

`tally = {A: 6, C: 4, D: 3}`

`loser = D`

Code Trace: `irv(ballots, cands)`



Round 3 begins: a zero counter for each remaining candidate.

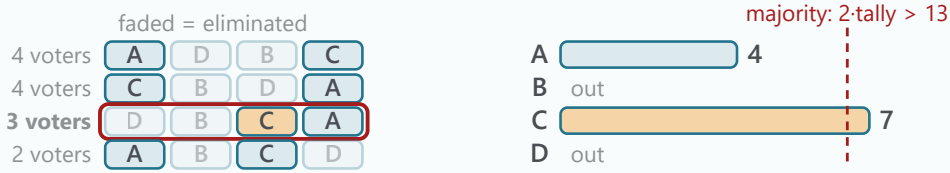

```
1  def irv(ballots, cands):
2    remaining = list(cands)
▶    while True:
▶      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

`remaining = {A, C}`

`tally = {A: 0, C: 0}`

Code Trace: `irv(ballots, cands)`



DBCA: top remaining candidate C ⇒ tally[C] += 3. (D is out, so these votes move to C.)

Step 14 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
▶      for count, order in ballots:
▶        top = next(a for a in order if a in remaining)
▶        tally[top] += count
8      leader = max(tally, key=tally.get)
9      if 2 * tally[leader] > sum(tally.values()):
10       return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

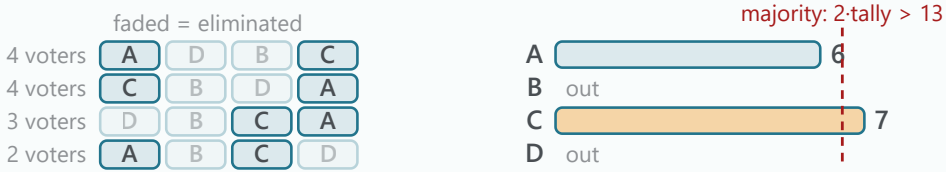
VARIABLES

`remaining = {A, C}` `count = 3`

`order = "DBCA"` `top = C`

`tally = {A: 4, C: 7}`

Code Trace: `irv(ballots, cands)`



leader = C: 2 × 7 = 14 > 13 ⇒ return **C**.

Step 15 / 15

```
1  def irv(ballots, cands):
2    remaining = list(cands)
3    while True:
4      tally = {a: 0 for a in remaining}
5      for count, order in ballots:
6        top = next(a for a in order if a in remaining)
7        tally[top] += count
▶      leader = max(tally, key=tally.get)
▶      if 2 * tally[leader] > sum(tally.values()):
▶        return leader
11     loser = min(tally, key=tally.get)
12     remaining.remove(loser)
```

VARIABLES

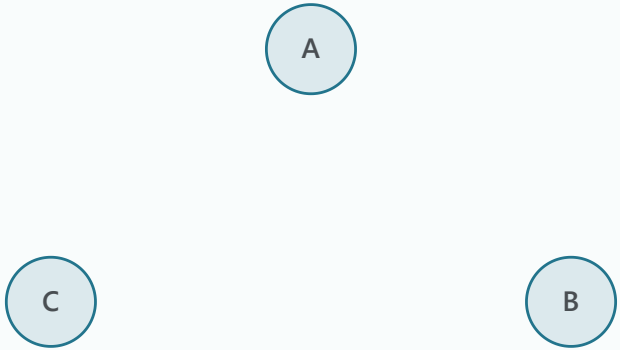
`remaining = {A, C}`

`tally = {A: 6, C: 7}`

`leader = C`

A Condorcet Cycle Step by Step

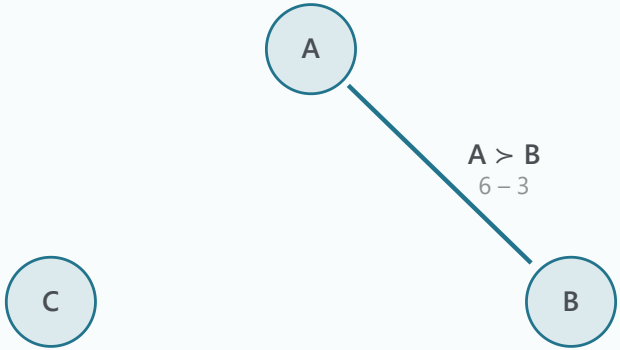
rank	4 voters	3 voters	2 voters
1.	A	B	C
2.	B	C	A
3.	C	A	B



9 voters, 3 candidates. Each voter's ranking on its own is consistent (transitive).

A Condorcet Cycle Step by Step

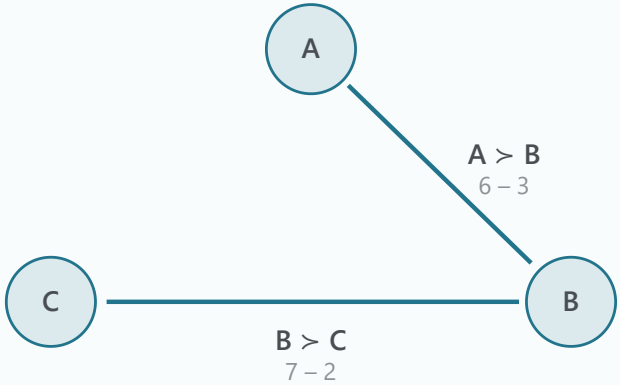
rank	4 voters	3 voters	2 voters
1.	**A**	**B**	C
2.	B	C	**A**
3.	C	A	B



A – B: 4 + 2 = 6 prefer A, 3 prefer B ⇒ majority **A ≻ B**.

A Condorcet Cycle Step by Step

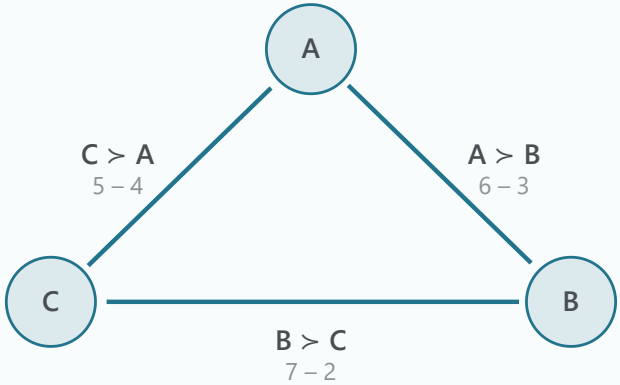
rank	4 voters	3 voters	2 voters
1.	A	**B**	**C**
2.	**B**	C	A
3.	C	A	B



B – C: 4 + 3 = 7 to 2 ⇒ majority **B** ≻ **C**. Transitivity would suggest A ≻ C.

A Condorcet Cycle Step by Step

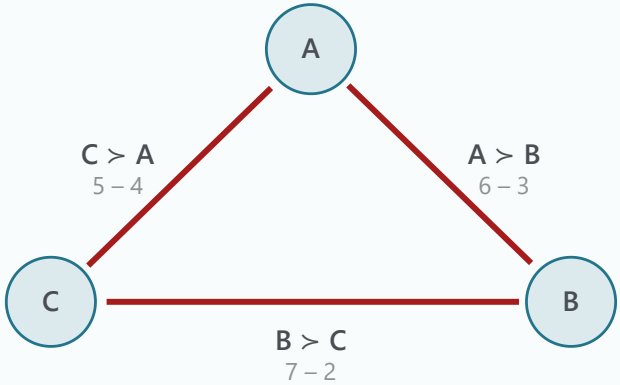
rank	4 voters	3 voters	2 voters
1.	**A**	B	**C**
2.	B	**C**	A
3.	C	A	B



C – A: 3 + 2 = 5 to 4 ⇒ majority **C** ≻ **A**!

A Condorcet Cycle Step by Step

rank	4 voters	3 voters	2 voters
1.	A	B	C
2.	B	C	A
3.	C	A	B



A ≻ B ≻ C ≻ A: the social preference is **cyclic**; every candidate is beaten by some majority, so there is no Condorcet winner.

The Agenda Decides the Winner

Sequential pairwise voting on the same cyclic profile: two candidates face off first, and the winner meets the next one.

Agenda 1: (A – B), then – C

- A – B: 6 – 3 ⇒ **A**
- A – C: 4 – 5 ⇒ **C**

Winner: **C**

A faces B first (A wins); A then meets C and loses: **C**.

Each of the three agendas yields a different winner: whoever sets the agenda (say, a committee chair) can pick the outcome. The candidate saved for last has the edge.

The Agenda Decides the Winner

Sequential pairwise voting on the same cyclic profile: two candidates face off first, and the winner meets the next one.

Agenda 1: (A – B), then – C

- A – B: 6 – 3 ⇒ **A**
- A – C: 4 – 5 ⇒ **C**

Winner: **C**

Agenda 2: (B – C), then – A

- B – C: 7 – 2 ⇒ **B**
- B – A: 3 – 6 ⇒ **A**

Winner: **A**

First B – C (B wins), then B – A: A wins 6–3: **A**.

Each of the three agendas yields a different winner: whoever sets the agenda (say, a committee chair) can pick the outcome. The candidate saved for last has the edge.

The Agenda Decides the Winner

Sequential pairwise voting on the same cyclic profile: two candidates face off first, and the winner meets the next one.

Agenda 1: (A – B), then – C

- A – B: 6 – 3 ⇒ **A**
- A – C: 4 – 5 ⇒ **C**

Winner: **C**

Agenda 2: (B – C), then – A

- B – C: 7 – 2 ⇒ **B**
- B – A: 3 – 6 ⇒ **A**

Winner: **A**

Agenda 3: (A – C), then – B

- A – C: 4 – 5 ⇒ **C**
- C – B: 2 – 7 ⇒ **B**

Winner: **B**

First A – C (C wins), then C – B: B wins 7–2: **B**.

Each of the three agendas yields a different winner: whoever sets the agenda (say, a committee chair) can pick the outcome. The candidate saved for last has the edge.

Back to the Warm-up: The Condorcet Paradox

The theory's answer: no

The warm-up profile (A≻B≻C, B≻C≻A, C≻A≻B): A–B 2–1, B–C 2–1, C–A 2–1. The social preference is cyclic; there is no Condorcet winner, and the agenda setter picks the outcome.

Agenda	First vote	Winner
(A–B), then C	A	**C**
(B–C), then A	B	**A**
(A–C), then B	C	**B**

How often?

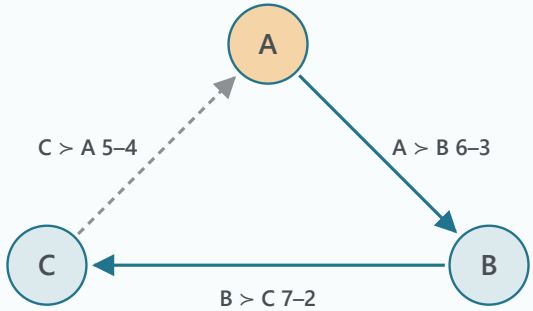
With three voters, three candidates and all rankings equally likely, 12 of the $6^3 = 216$ profiles are cycles: 5.6%. As the number of voters grows, this rises to about 8.8% (Guilbaud, 1952).

In practice: Election and survey data show cycles even less often than this "random preferences" estimate; real preferences are not random but similar to one another (Regenwetter et al., 2006).

In a cycle the agenda setter decides the winner: each of the three possible agendas elects a different candidate, and the one saved for last has the edge.

The Kemeny Rule: The Most Consistent Ranking

- Kemeny (1959): choose the social ranking that **agrees most** with the voters' pairwise preferences.
- The score of a ranking: for every pair where x comes before y, add the number of voters who prefer x to y.
- If a Condorcet winner exists, it goes first: on the main profile D ≻ B ≻ C ≻ A (44).
- It also decides cycles: it breaks the **weakest** majority ⇒ A ≻ B ≻ C (17).
- Finding the best ranking is **NP-hard** (Bartholdi, Tovey, Trick, 1989).



dashed: the weakest majority Kemeny overrides (5–4)

Applications: rank aggregation (Dwork et al., 2001): combining search engine results, ranking contestants and committee rankings.

Code Trace: `kemeny(ballots, cands)`

A ≻ B ≻ C · B ≻ C ≻ A ·
A ≻ C ≻ B · C ≻ A ≻ B ·
B ≻ A ≻ C · C ≻ B ≻ A ·

rankings tried (agreement score; orange = best)

ballots = [(4, "ABC"), (3, "BCA"), (2, "CAB")] (the cyclic profile). No best ranking yet; score −1.

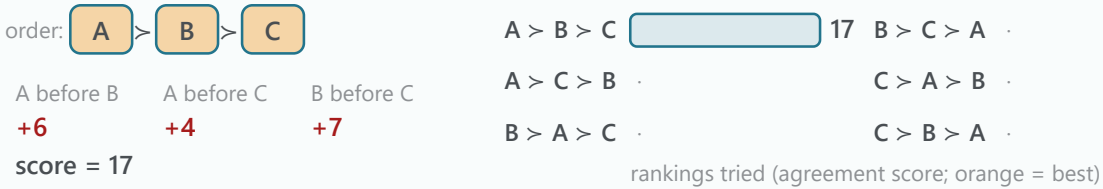
Step 1 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
▶    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

VARIABLES

`best = None` `best_score = -1`

Code Trace: `kemeny(ballots, cands)`



order = A ≻ B ≻ C: agreeing voters over the 3 pairs 6 + 4 + 7 = **17**.

Step 2 / 14

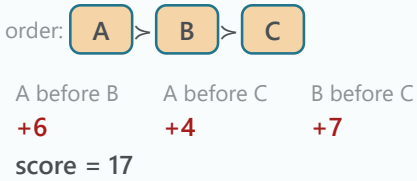
```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
▶    for order in permutations(cands):
▶      score = 0
▶      for i, x in enumerate(order):
▶        for y in order[i + 1:]:
▶          score += sum(n for n, b in ballots
▶                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

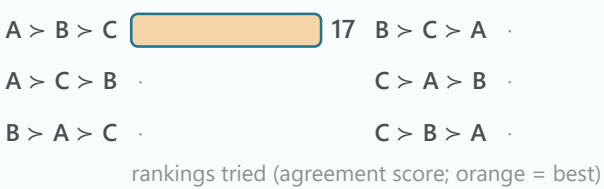
VARIABLES

`order = (A, B, C)` `score = 17`

`best = None` `best_score = -1`

Code Trace: `kemeny(ballots, cands)`





17 > previous best ⇒ best = A ≻ B ≻ C.

Step 3 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
▶      if score > best_score:
▶        best, best_score = order, score
12   return best, best_score
```

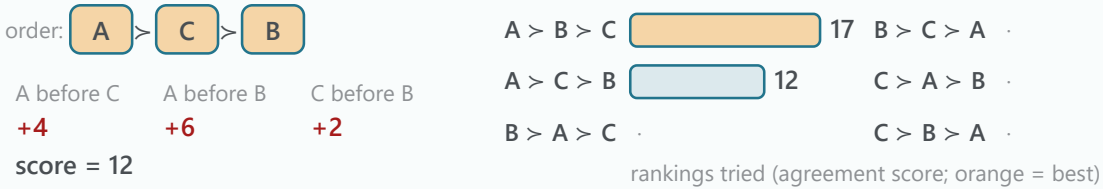
VARIABLES

`order = (A, B, C)` `score = 17`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



order = A ≻ C ≻ B: agreeing voters over the 3 pairs 4 + 6 + 2 = **12**.

Step 4 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
▶    for order in permutations(cands):
▶      score = 0
▶      for i, x in enumerate(order):
▶        for y in order[i + 1:]:
▶          score += sum(n for n, b in ballots
▶                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

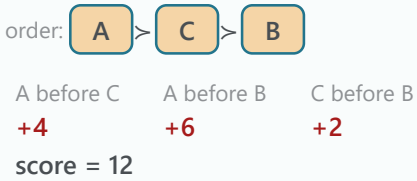
VARIABLES

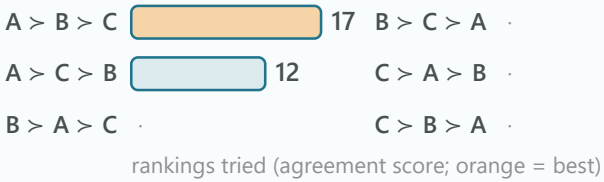
`order = (A, C, B)` `score = 12`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`





12 ≤ 17: best unchanged.

Step 5 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
▶      if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

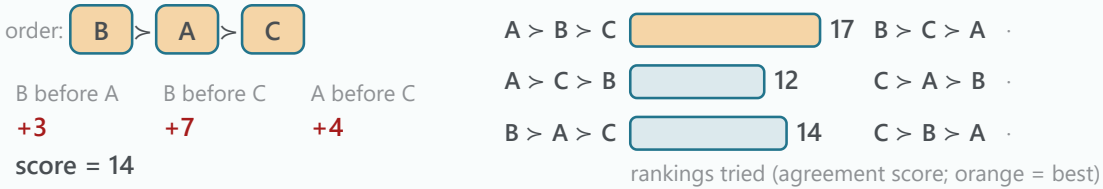
VARIABLES

`order = (A, C, B)` `score = 12`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



order = B ≻ A ≻ C: agreeing voters over the 3 pairs 3 + 7 + 4 = **14**.

Step 6 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
▶    for order in permutations(cands):
▶      score = 0
▶      for i, x in enumerate(order):
▶        for y in order[i + 1:]:
▶          score += sum(n for n, b in ballots
▶                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

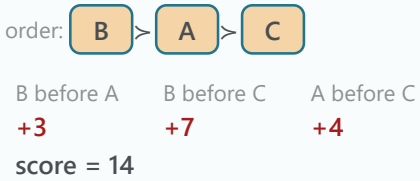
VARIABLES

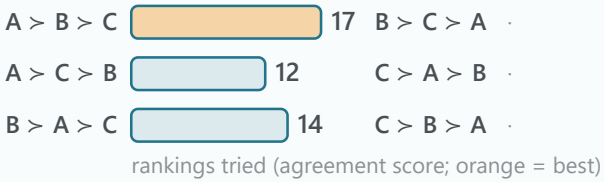
`order = (B, A, C)` `score = 14`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`





14 ≤ 17: best unchanged.

Step 7 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
▶      if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

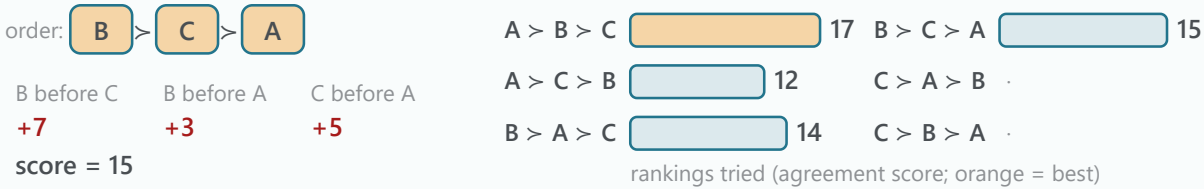
VARIABLES

`order = (B, A, C)` `score = 14`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



order = B ≻ C ≻ A: agreeing voters over the 3 pairs 7 + 3 + 5 = **15**.

Step 8 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
▶    for order in permutations(cands):
▶      score = 0
▶      for i, x in enumerate(order):
▶        for y in order[i + 1:]:
▶          score += sum(n for n, b in ballots
▶                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

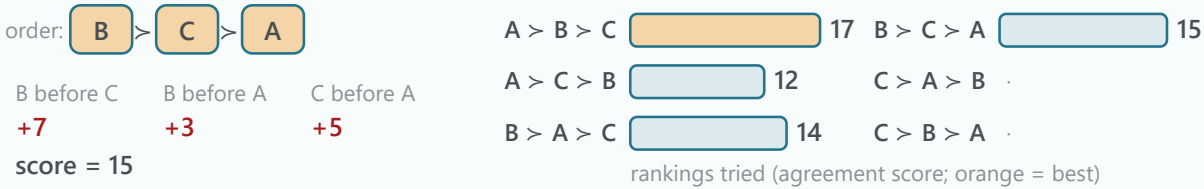
VARIABLES

`order = (B, C, A)` `score = 15`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



15 ≤ 17: best unchanged.

Step 9 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
▶      if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

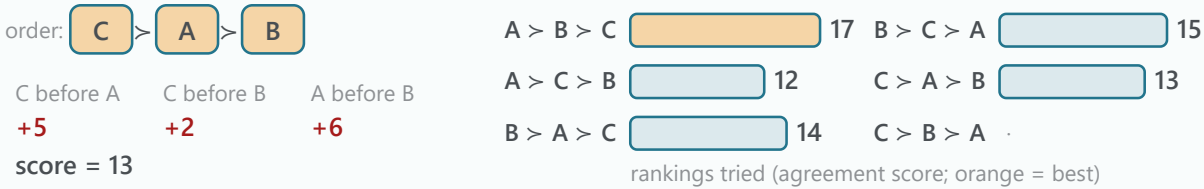
VARIABLES

`order = (B, C, A)` `score = 15`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



order = C ≻ A ≻ B: agreeing voters over the 3 pairs 5 + 2 + 6 = **13**.

Step 10 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
▶    for order in permutations(cands):
▶      score = 0
▶      for i, x in enumerate(order):
▶        for y in order[i + 1:]:
▶          score += sum(n for n, b in ballots
▶                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

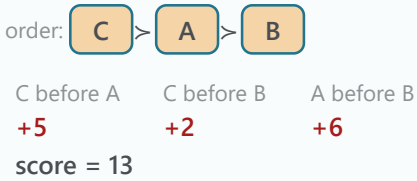
VARIABLES

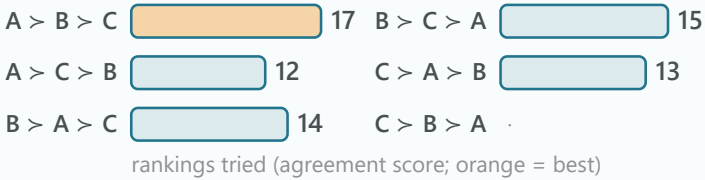
`order = (C, A, B)` `score = 13`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`





rankings tried (agreement score; orange = best)

13 ≤ 17: best unchanged.

Step 11 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
▶      if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

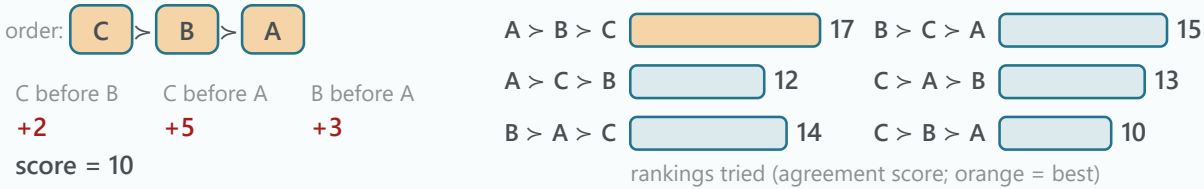
VARIABLES

`order = (C, A, B)` `score = 13`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



order = C ≻ B ≻ A: agreeing voters over the 3 pairs 2 + 5 + 3 = **10**.

Step 12 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
▶    for order in permutations(cands):
▶      score = 0
▶      for i, x in enumerate(order):
▶        for y in order[i + 1:]:
▶          score += sum(n for n, b in ballots
▶                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

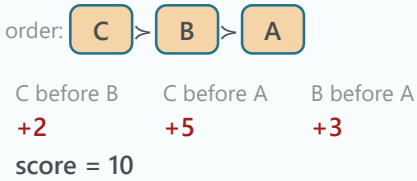
VARIABLES

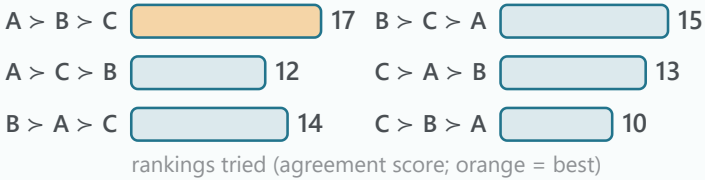
`order = (C, B, A)` `score = 10`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`





10 ≤ 17: best unchanged.

Step 13 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
▶      if score > best_score:
11       best, best_score = order, score
12   return best, best_score
```

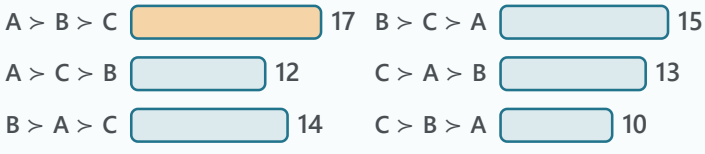
VARIABLES

`order = (C, B, A)` `score = 10`

`best = (A, B, C)`

`best_score = 17`

Code Trace: `kemeny(ballots, cands)`



rankings tried (agreement score; orange = best)

Result A ≻ B ≻ C (17): only the weakest majority, C ≻ A (5–4), is overridden. With m candidates, all m! rankings are tried.

Step 14 / 14

```
1  from itertools import permutations
2  def kemeny(ballots, cands):
3    best, best_score = None, -1
4    for order in permutations(cands):
5      score = 0
6      for i, x in enumerate(order):
7        for y in order[i + 1:]:
8          score += sum(n for n, b in ballots
9                       if b.index(x) < b.index(y))
10     if score > best_score:
11       best, best_score = order, score
▶    return best, best_score
```

VARIABLES

`best = (A, B, C)`

`best_score = 17`

Which Criteria Does Each Rule Satisfy?

Rule	Elects the Condorcet winner	Never elects the Condorcet loser	Monotone	Strategy proof
Plurality	no	no (A)	yes	no
Borda	no (B)	yes	yes	no
Instant-runoff	no (C)	yes	no	no
Approval	no	no	yes	no
Condorcet (Copeland)	yes	yes	yes	no
Kemeny	yes	yes	yes	no

- **Monotonicity:** raising the winner on a ballot should never make it lose. IRV can violate this: moving a candidate up changes the elimination order.
- In a cycle there is no Condorcet winner, so the bare rule cannot decide and needs an extension. The **maximin** (Simpson) rule takes each candidate's worst pairwise vote count; the highest wins: on the sample profile A 6, B 6, C 4, D 7 → D. It elects the Condorcet winner when one exists and still decides cycles.
- Strategyproof: no voter can get a better outcome by misreporting their ranking. The "no"s in the last column come from a general theorem (Gibbard–Satterthwaite).

IRV Violates Monotonicity

rank	6 voters	6 voters	5 voters
1.	**A**	**B**	**C**
2.	B	C	A
3.	C	A	B

candidate	round 1
A	6
B	6
C	5

17 voters, 9 votes for a majority. Round 1: A = 6, B = 6, C = 5 ⇒ no majority; **C** has the fewest and is eliminated.

IRV Violates Monotonicity

rank	6 voters	6 voters	5 voters
1.	**A**	**B**	∈
2.	B	C	**A**
3.	C	A	B

candidate	round 2
A	**11**
B	6
C	—

C's 5 votes go to A: A = 11, B = 6 ⇒ **A wins**.

IRV Violates Monotonicity

rank	6+2 voters	4 voters	5 voters
1.	**A**	B	C
2.	B	C	A
3.	C	A	B

candidate	—
A	?
B	?
C	?

Before the election A campaigns: **2** of the 6 B ≻ C ≻ A voters move A to the top (A ≻ B ≻ C). A's support only **grew**.

IRV Violates Monotonicity

rank	6+2 voters	4 voters	5 voters
1.	**A**	**B**	**C**
2.	B	C	A
3.	C	A	B

candidate	round 1
A	8
B	4
C	5

New round 1: A = 8, B = 4, C = 5 ⇒ now **B** has the fewest and is eliminated (not C).

IRV Violates Monotonicity

rank	6+2 voters	4 voters	5 voters
1.	**A**	~~B~~	**C**
2.	B	**C**	A
3.	C	A	B

candidate	round 2
A	8
~~B~~	—
C	**9**

B's 4 votes go to C: C = 9, A = 8 ⇒ **C wins**. A lost because it gained support: IRV is **not monotone**.

Tactical Voting under Plurality (Vote Splitting)

rank	4 voters	4 voters	3 voters	2 voters
1.	**A**	**C**	**D**	**A**
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	plurality
A	**6**
B	0
C	4
D	3

With sincere votes, A wins plurality. For group 3 ($D \succ B \succ C \succ A$), A is the **worst** candidate!

Step 1 / 3

"Don't waste your vote": voters back a lesser evil who can win instead of their favorite (Duverger's law: plurality pushes toward two parties).

Tactical Voting under Plurality (Vote Splitting)

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	**D**	A
2.	D	B	B	B
3.	B	D	**C**	C
4.	C	A	A	D

candidate	plurality
A	**6**
B	0
C	4
D	3

Group 3's favorite D (3 votes) cannot win. The candidate who can stop A is C, and this group prefers C to A.

"Don't waste your vote": voters back a lesser evil who can win instead of their favorite (Duverger's law: plurality pushes toward two parties).

Tactical Voting under Plurality (Vote Splitting)

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	**C**	A
2.	D	B	D	B
3.	B	D	B	C
4.	C	A	A	D

candidate	plurality
A	6
B	0
C	**7**
D	0

Group 3 votes for C (ballot $C \succ D \succ B \succ A$): C = 7, A = 6 $\Rightarrow$ **C** wins, which this group likes better than A.

"Don't waste your vote": voters back a lesser evil who can win instead of their favorite (Duverger's law: plurality pushes toward two parties).

Strategic Voting under Borda Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	sincere
A	18
B	**22**
C	17
D	21

Sincere Borda count: B = 22, D = 21. Group 3 ($D \succ B \succ C \succ A$) sees its favorite D losing by 1 point.

Strategic Voting under Borda Step by Step

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	**C**	B
3.	B	D	**B**	C
4.	C	A	A	D

candidate	sincere
A	18
B	**22**
C	17
D	21

Manipulation by burying: the 3 voters push the main rival B from 2nd to 3rd place; ballot D ≻ C ≻ B ≻ A.

Strategic Voting under Borda Step by Step

rank	4 voters	4 voters	3 voters	2 voters	candidate	sincere	change
1.	A	C	D	A	**A**	18	0
2.	D	B	**C**	B	**B**	22	−3
3.	B	D	**B**	C	**C**	17	+3
4.	C	A	A	D	**D**	21	0

For each manipulator, B gets 1 point instead of 2 and C gets 2 instead of 1: B −3, C +3.

Strategic Voting under Borda Step by Step

rank	4 voters	4 voters	3 voters	2 voters	candidate	sincere	strategic
1.	A	C	D	A	**A**	18	18
2.	D	B	C	B	**B**	22	19
3.	B	D	B	C	**C**	17	20
4.	C	A	A	D	**D**	**21**	**21**

New totals: D = 21, C = 20, B = 19, A = 18 ⇒ **D** wins. Group 3 got its favorite elected by lying.

How Much Damage Does Strategic Voting Do?

Plurality

Vote splitting: two similar candidates share votes and a third one wins. Tactical voting is common and simple.

Borda

Burying a rival at the bottom is very effective; if everyone buries, the ranking drifts away from voters' true preferences.

Instant-runoff

Manipulation is harder to compute but **possible**, and the rule is not monotone. "IRV is immune to strategic voting" is a common myth.

Computational barrier

For some rules (e.g. IRV and a variant of Copeland) finding a profitable manipulation is NP-hard (Bartholdi et al., 1989–91).

The effect depends on the preference distribution: in close races strategic voting can change the winner. Transparent rules and clear information reduce some manipulation, but no general rule eliminates strategic behavior in every case.

Arrow's Impossibility Theorem: The Conditions

Reasonable-looking conditions for a rule that outputs a social **ranking** (a social welfare function):

Unrestricted domain
Every preference profile is allowed; the output is a complete, transitive ranking.

Pareto (unanimity)
If everyone prefers x to y, so does society.

Independence of irrelevant alternatives (IIA)
The social order of x and y depends only on how voters rank x versus y.

Non-dictatorship
No single voter's preference is society's preference in every profile.

Arrow (1951): With at least **3 candidates**, no rule satisfies all four conditions. (With 2 candidates, simple majority satisfies them all: May's theorem.)

Borda and Irrelevant Alternatives

rank	4 voters	4 voters	3 voters	2 voters
1.	A	C	D	A
2.	D	B	B	B
3.	B	D	C	C
4.	C	A	A	D

candidate	4 candidates
A	18
B	**22**
C	17
D	**21**

Borda with 4 candidates: B = 22 > D = 21; society ranks **B above D**.

Borda and Irrelevant Alternatives

rank	4 voters	4 voters	3 voters	2 voters
1.	A	~~C~~	D	A
2.	D	B	B	B
3.	B	D	~~C~~	~~C~~
4.	~~C~~	A	A	D

candidate	4 candidates
A	18
B	**22**
C	17
D	**21**

Now let C drop out. C was neither winning nor second: an "irrelevant" candidate for the B–D order.

Borda and Irrelevant Alternatives

rank	4 voters	4 voters	3 voters	2 voters
1.	A	B	D	A
2.	D	D	B	B
3.	B	A	A	D

candidate	4 candidates	no C
A	18	12
B	**22**	**13**
C	17	
D	**21**	**14**

New Borda (2, 1, 0): D = 14 > B = 13. No voter changed their B–D order (6 prefer B, 7 prefer D), yet the social order flipped: Borda **violates IIA**. Arrow's IIA can fail even with a fixed candidate set: if the 4-voter A ≻ D ≻ B ≻ C group only swaps C and B (A ≻ D ≻ C ≻ B), B drops from 22 to 18 while D stays at 21 (C rises to 21): the B–D order flips again.

What Arrow's Theorem Means in Practice

What it says

No collective decision rule meets every reasonable criterion at once; every rule gives up some condition.

What it does not say

It does not say voting is meaningless, nor does it rule out any particular rule.

Who gives up what?

Borda: IIA. Condorcet/majority ranking: transitivity (cycles). Dictatorship: fairness.

For institutions

Decide explicitly which criterion takes priority, and document it.

The way out is to relax a condition: restrict the preference domain (single-peaked preferences), or use intensity information (scores, money) instead of rankings.

The Gibbard–Satterthwaite Theorem

Theorem (Gibbard 1973, Satterthwaite 1975): With at least **3 candidates** and every ranking allowed (unrestricted domain), every rule that picks a single winner in every profile, lets every candidate win (onto) and is **strategyproof** (nobody gains by lying) is a **dictatorship**.

Evidence in our profile

Group 3 voting for C under plurality; burying B under Borda.

Relation to Arrow

Arrow is about rankings, G–S about a single winner; G–S can be derived from Arrow.

Mechanism design

Without money there is no general incentive-compatible rule. With money, VCG works.

- The design goal is not to eliminate strategic behavior but to **limit** it: less information, computational hardness, a restricted preference domain.

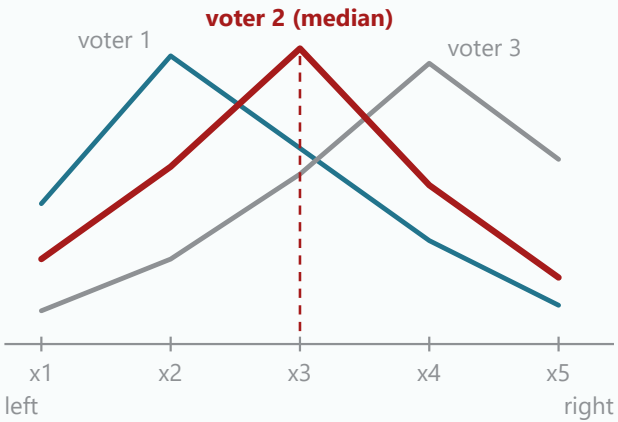
May's Theorem: Simple Majority for Two Candidates

With two candidates (x, y), which rule is "fair"? May (1952) showed that **only simple majority** satisfies three conditions.

Condition	Meaning	Violated by
Anonymity	Voter names don't matter; permuting the ballots leaves the outcome unchanged	Dictatorship; weighted voting
Neutrality	Candidate names don't matter; both candidates are treated symmetrically	"The status quo stays without ⅔" (supermajority)
Positive responsiveness	If x is tied or winning and one voter switches to x, x wins outright	"Always flip a coin"; "odd vote count wins"

With two candidates the debate over rules ends; the trouble (cycles, Arrow, G–S) starts with **three or more** candidates.

A Way Out: Single-Peaked Preferences



- Preferences are **single-peaked** if candidates lie on an axis (left–right, budget size, location), each voter has a favorite position (peak), and utility falls as a candidate moves away from the peak.
- **Median voter theorem** (Black, 1948): with an odd number of voters, the median voter's peak is the Condorcet winner; no cycles arise.
- "Report your peak, pick the median" is **strategyproof**: G–S no longer applies once the domain is restricted.

The median mechanism for facility location is an application of this idea.

Four Results Side by Side

Theorem	Assumptions	Conclusion	Way out
Arrow (1951)	≥ 3 candidates, unrestricted domain, Pareto, IIA	The rule is a dictatorship	Restrict the domain (single-peaked); drop IIA; cardinal information (scores, intensity) instead of rankings
Gibbard–Satterthwaite (1973, 1975)	≥ 3 candidates, unrestricted domain, onto (any candidate can win), strategyproof	The rule is a dictatorship	Single-peaked preferences; randomized rules (e.g. random dictator); money (VCG)
May (1952)	2 candidates; anonymity, neutrality, positive responsiveness	The rule is simple majority (and conversely)	None needed: not an impossibility but a **characterization**
Single-peaked preferences (Black, 1948)	Candidates on an axis, each voter has one peak, odd number of voters	The median voter's peak is the Condorcet winner; the median rule is strategyproof	It is itself a way out: it restricts the domain

In Arrow and G–S, dictatorship follows only when all assumptions are demanded together; relax one and positive results become possible.

Computational Social Choice

Where computer science, economics and social choice meet: rules are judged not only by the criteria they satisfy but also by their **computational cost**.

Computing the winner

With n voters and m candidates, plurality and Borda take $O(nm)$ time, Copeland $O(nm^2)$. For Kemeny and Dodgson (the winner is the candidate needing the fewest adjacent swaps to become a Condorcet winner), finding the winner is NP-hard (Bartholdi, Tovey, Trick, 1989).

Computing manipulations

A single manipulator finds a profitable ballot under plurality and Borda with a polynomial-time greedy algorithm; under IRV the problem is NP-hard. Hardness is a “barrier”, but only in the worst case.

Preference elicitation

A full ranking of m candidates carries $\log_2(m!)$ bits (≈ 22 bits for $m = 10$). Asking voters questions step by step and collecting only enough to settle the winner cuts communication.

Fair division and judgment aggregation

Two more topics in this field.

COMPUTATIONAL SOCIAL CHOICE

Judgment Aggregation: The Doctrinal Paradox

A three-judge court: the defendant pays damages only if **p** (the contract was valid) **and q** (the contract was breached).

Judge	p	q	Verdict: p ∧ q (p and q)
Judge 1	yes	yes	yes
Judge 2	yes	no	no
Judge 3	no	yes	no
Majority	**yes (2–1)**	**yes (2–1)**	**no (1–2)**

- Each judge is consistent, but voting premise by premise the majority says “p true, q true, p ∧ q false”: an **inconsistent** collective judgment (Kornhauser and Sager, 1986).
- Decide by the premises and the defendant pays; decide by the conclusion and they don't: the outcome again depends on **procedure**.
- List and Pettit (2002): no aggregation rule is anonymous, systematic and always consistent; Arrow's theorem for judgments.

SUMMARY

Voting Rules in the Real World

Two-round system
Presidential elections in Türkiye and France: without a first-round majority, the top two face a runoff.

Instant-runoff
Australian House of Representatives, some US cities; voters submit full rankings.

Plurality
Single-member districts in the UK and the US.

Borda-like scoring
Eurovision (12, 10, 8, …), sports awards (e.g. most valuable player votes).

Approval
Some scientific society elections; "times that work for me" polls.

Online
Reddit and forum votes, community polls; class and club decisions.

SUMMARY

Multiple Seats: Proportional Representation and D'Hondt

- S seats instead of one winner; seat shares should track vote shares (**proportional representation**).
- **D'Hondt**: divide each party's votes by 1, 2, 3, …; the S largest quotients win. Türkiye uses D'Hondt per district, plus a national threshold.
- Example: 6 seats; A 42,000, B 31,000, C 15,000, D 12,000 votes → **A 3, B 2, C 1, D 0**.
- A gets 50% of seats with 42% of votes; D gets none with 12%: D'Hondt favors large parties. Largest remainder (Hare): integer part of share × 6, leftovers to the largest fractions: A 2, B 2, C 1, D 1.

Divisor	A	B	C	D
÷1	**42,000**	**31,000**	**15,000**	12,000
÷2	**21,000**	**15,500**	7,500	6,000
÷3	**14,000**	10,333	5,000	4,000
÷4	10,500	7,750	3,750	3,000
Seats	**3**	**2**	**1**	**0**

Highlighted cells: the six largest quotients.

Every method is a trade-off: D'Hondt favors stable governments, largest remainder small parties.

SUMMARY

Which Voting Rule Can Be Called Fair?

- Fairness depends on criteria such as representation, majority support, protection of minorities and strategyproofness.
- **Plurality** is simple and fast, but can split votes and elect a winner without majority support (A: 6/13).
- **Borda** uses more preference information and favors the compromise candidate (B), but is open to strategic ranking.
- **Instant-runoff** seeks majority support, but is neither monotone nor strategyproof.
- **Condorcet methods** respect pairwise majorities, but a Condorcet winner does not always exist and cycles need an extension rule.

Choose the rule by the context and the criteria that matter most: quick plurality may suit a club presidency, a ranked rule may be fairer for a class event.

Summary

Rules

Majority, plurality, Borda, approval, instant-runoff, Condorcet, Kemeny: one profile, 4 different winners.

Condorcet paradox

Transitive individual preferences can produce a cyclic social preference; the agenda setter picks the outcome. Kemeny picks the ranking that disagrees least with the majorities; IRV is not monotone.

Strategic voting

Tactical voting under plurality, burying under Borda: honesty is not the best strategy under every rule.

Arrow

With ≥ 3 candidates, unrestricted domain + Pareto + IIA + non-dictatorship cannot all hold.

Gibbard–Satterthwaite

With ≥ 3 candidates, a strategyproof onto rule = dictatorship.

May and single-peakedness

With 2 candidates, simple majority is the only fair rule; with single-peaked preferences the median is strategyproof.

SUMMARY

Sample Problems

1. 9 voters: 4 × A ≻ B ≻ C, 3 × C ≻ B ≻ A, 2 × B ≻ C ≻ A. Who wins under plurality, IRV, Borda and Condorcet?
2. In problem 1, what do the A voters achieve under Borda by burying B (A ≻ C ≻ B)?
3. On the cyclic profile (4 ABC, 3 BCA, 2 CAB), what agenda makes B win?
4. What is the Kemeny ranking of the profile in problem 1? (Compare the scores of all 6 rankings.)
5. "The status quo changes only with ⅔ of the vote": which of May's conditions does this rule violate?
6. 11 voters: 2 × ACB, 3 × BAC, 2 × BCA, 4 × CBA. Who is the Condorcet winner? Which rule $s = (1, a, 0)$ ($0 \le a \le 1$) elects them?

▶ **Show answers**

SUMMARY

References

- F. Brandt, V. Conitzer, U. Endriss, J. Lang, A. D. Procaccia (eds.), *Handbook of Computational Social Choice*, Cambridge University Press, 2016.
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (eds.), *Algorithmic Game Theory*, Cambridge University Press, 2007 (Chapter 9).
- K. J. Arrow, *Social Choice and Individual Values*, Wiley, 1951.
- A. Gibbard, "Manipulation of voting schemes: a general result", *Econometrica* 41(4), 1973.
- M. A. Satterthwaite, "Strategy-proofness and Arrow's conditions", *J. Economic Theory* 10, 1975.
- K. O. May, "A set of independent necessary and sufficient conditions for simple majority decision", *Econometrica* 20(4), 1952.
- D. Black, "On the rationale of group decision-making", *J. Political Economy* 56(1), 1948.
- J. J. Bartholdi, C. A. Tovey, M. A. Trick, "The computational difficulty of manipulating an election" and "Voting schemes for which it can be difficult to tell who won the election", *Social Choice and Welfare* 6, 1989 · M. Balinski, H. P. Young, *Fair Representation*, Yale University Press, 1982 (proportional representation).
- J. G. Kemeny, "Mathematics without numbers", *Daedalus* 88(4), 1959; C. Dwork, R. Kumar, M. Naor, D. Sivakumar, "Rank aggregation methods for the web", WWW 2001.
- A. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017 (chapter on social choice and voting); D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, Cambridge University Press, 2010 (Chapter 23: Voting).
- Course pages: **A. Procaccia, 15-896 (CMU)** · **N. Shah, CSC304 (Toronto)**.

The End

Chapter 7: Social Choice and Voting

Previous: Chapter 6: Online Platforms · Next: Chapter 8: Network Design and Routing