Chapter 6: Online Platforms

Algorithmic Game Theory

Two-sided markets | Network effects and critical mass | Platform pricing | Sponsored search: GSP and VCG | Reputation systems

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_06_Online_Platforms.html

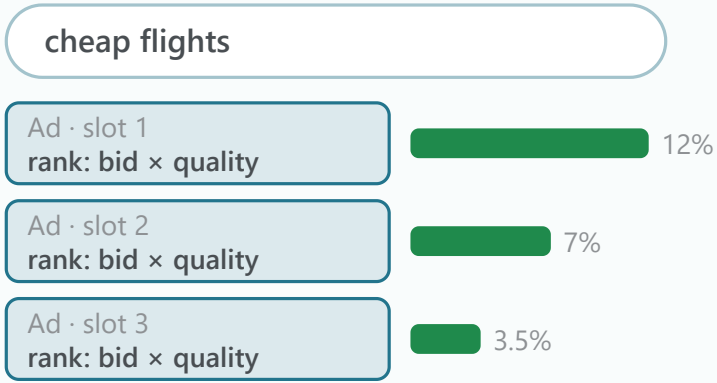
Warm-up Game: The Sponsored Search Auction

The rules

Advertisers bid per click; ads are ranked into three slots by bid × quality score, and higher slots get more clicks.

Think about it

- If a click earns you 8 TL, is bidding 8 TL best?
- Does the highest bidder always get the top slot?



Bar: the slot's click-through rate (Search Ads scenario)

Learning Outcomes

Modeling platforms

Describe two-sided markets, cross-side network effects, and the platform as a mechanism designer.

Computing critical mass

Find the equilibria, the threshold and the "tipping" dynamics in a simple adoption model.

Explaining price structure

Show in a two-sided numerical example which side is subsidized, and why.

Ad auctions

Compute GSP and VCG payments from the same bids; show why truthful bidding is not an equilibrium of GSP.

Measuring reputation

Rank with Bayesian averages; compute when fake reviews pay off.

Discussing competition

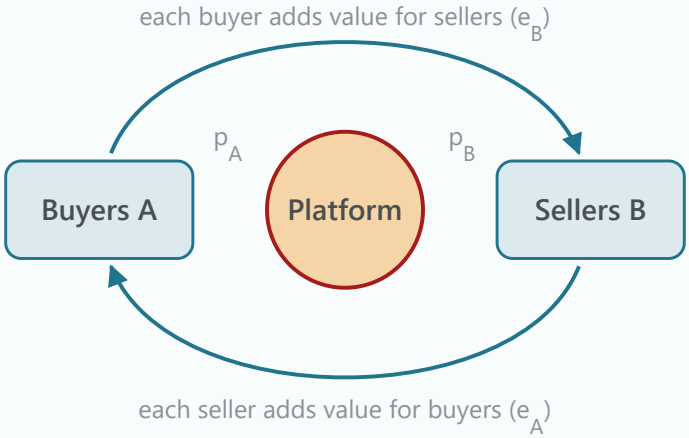
Interpret how tipping, switching costs, multi-homing and regulation (the EU Digital Markets Act, DMA) affect platform competition.

What Is an Online Platform?

- An online platform: a digital ecosystem (an intermediary) that brings users, sellers and service providers together.
- The value comes less from the platform itself than from the sides' **access to each other**: a two-sided market.
- The platform sets the rules: who joins, who sees whom, who pays what → a **mechanism**.

Type	Side 1	Side 2	Examples
E-commerce marketplace	buyers	sellers	Trendyol, Hepsiburada, Amazon
Sharing economy	guests / riders	hosts / drivers	Airbnb, BlaBlaCar, Uber
Social media	users	advertisers, creators	Instagram, TikTok, Facebook
Search engine	searchers	advertisers	Google, Bing

Two-Sided Markets and Cross-Side Network Effects



Cross-side network effect

The number of participants on one side determines the value the other side gets (more sellers → more choice for buyers).

Price structure

The platform chooses not only the total price but how the fee is split between the sides (p_A, p_B).

The chicken-and-egg problem

No sellers, no buyers; no buyers, no sellers: one side has to be “pulled in” first.

The Platform Is a Mechanism, Users Are Players

- Platform design (fees, ranking and rating rules) steers users' strategic behavior.
- Users choose strategies according to incentives; the resulting behavior is a **Nash equilibrium** under the rules.

The platform's incentives

+ Bonuses: extra pay for drivers in busy areas (Uber)
+ Discounts, coupons, free trials
+ Ranking and recommendation algorithms → visibility
+ Ratings and badges (trusted seller)

Users' strategic responses

– Fake reviews and rating inflation
– Multiple accounts, bonus abuse
– Gaming the algorithm (click farms)
– Retaliatory negative reviews

Right incentives → cooperation and efficiency. Wrong incentives → manipulation and loss of trust.

Types of Network Effects

Direct (same side)

WhatsApp: the more of your friends use it, the more valuable it is to you.

Indirect / cross-side

Uber: more drivers → shorter waits → more riders → more work for drivers.

Negative

Crowding → congestion, information overload; sellers on the same side compete with each other.

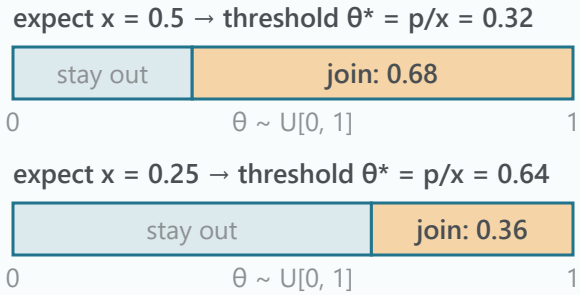
Local

Value comes from your neighborhood (city, circle of friends), not from all users; competition is won locally.

A positive network effect creates a **feedback loop**: winners grow, losers shrink. The market can "tip" one way.

A Simple Adoption Model

- N = 1000 potential users; user i's type $\theta_i \sim U[0, 1]$ measures how much they care about the platform.
- If a fraction x of the population joins, i's utility is $\theta_i \cdot x - p$, where p = 0.16 (a fee or the effort of joining).
- A user joins $\Leftrightarrow \theta_i \cdot x \geq p \Leftrightarrow \theta_i \geq p/x$. Since θ is uniform on [0, 1], the fraction with θ above p/x is $1 - p/x$.

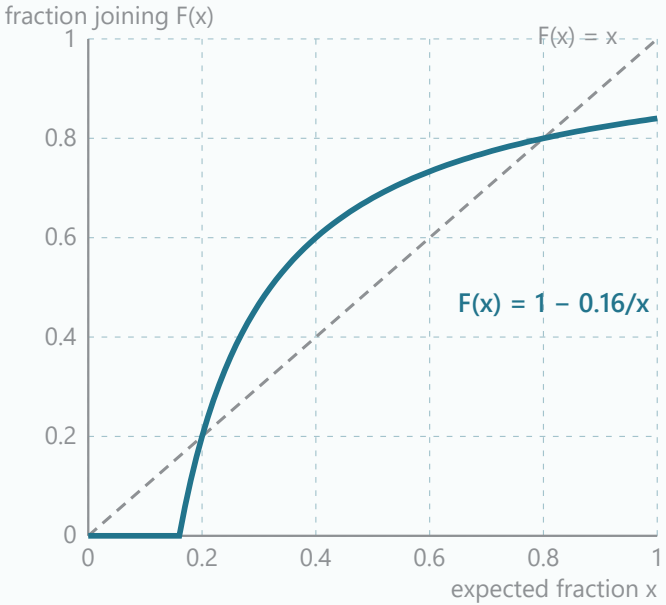


Fraction joining if x is expected: $F(x) = 1 - p/x$ (0 if $x \leq p$)

- **Equilibrium**: a self-fulfilling expectation, F(x) = x.
- **Dynamics**: this period's adopters set next period's expectation, $x_{t+1} = F(x_t)$.

A simplified version of the network-effects model in Easley and Kleinberg (2010, Chapter 17).

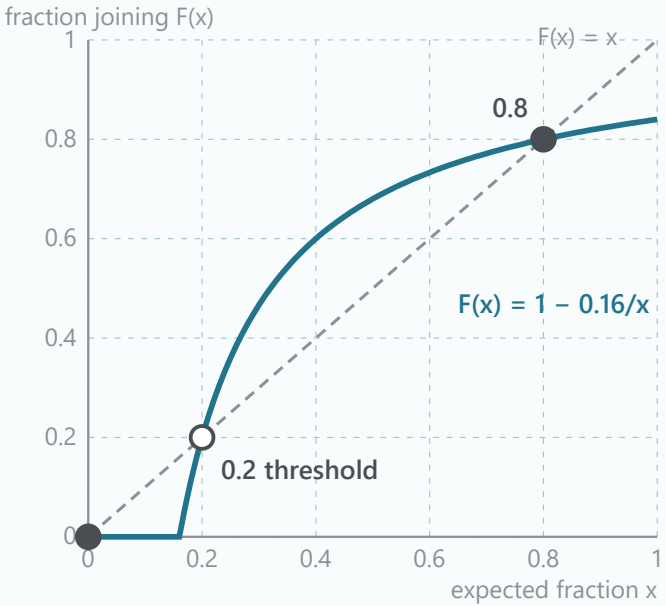
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0		
1		
2		
3		
4		
5		
6		

The curve F(x) = 1 − 0.16/x: the fraction who actually join when “a fraction x will join” is expected. Nobody joins while x ≤ 0.16.

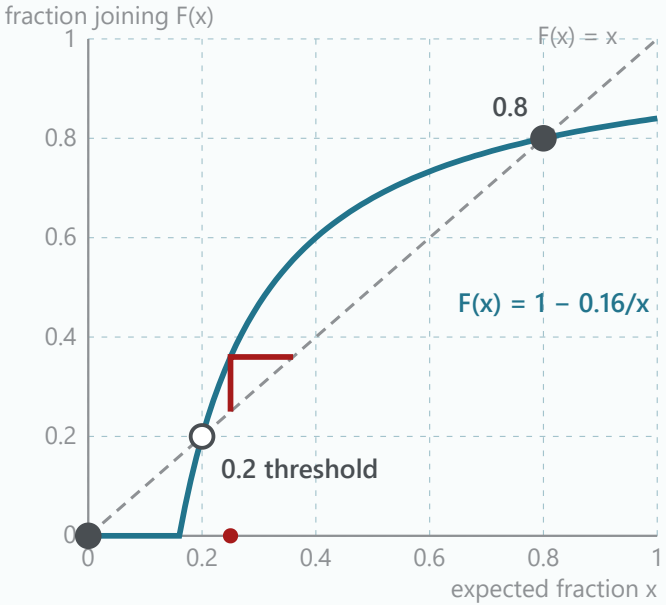
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0		
1		
2		
3		
4		
5		
6		

Equilibria satisfy F(x) = x: multiplying $1 - 0.16/x = x$ by x gives $x^2 - x + 0.16 = 0 \rightarrow$ **x = 0.2 and x = 0.8**; also x = 0 (nobody). The hollow point is an unstable equilibrium: a small deviation pushes the dynamics away from it.

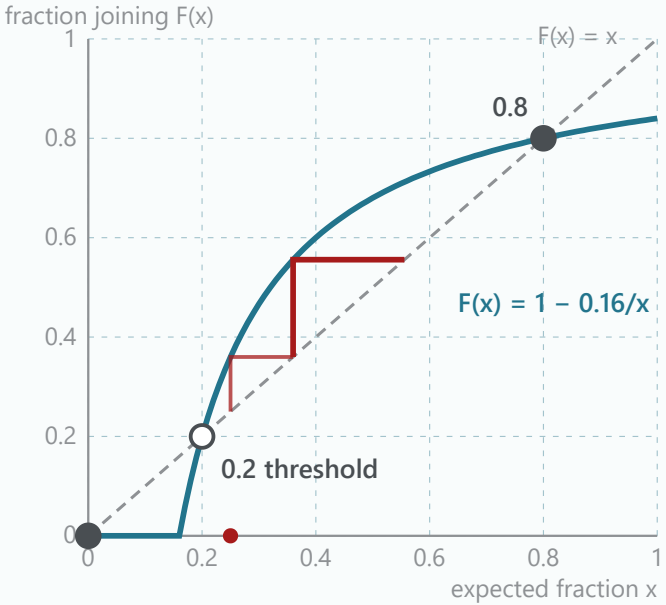
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	
1	0.36	
2		
3		
4		
5		
6		

250 users are expected ($x_0 = 0.25$): those with $\theta \geq 0.16/0.25 = 0.64$ join → $x_1 =$ **0.36**.

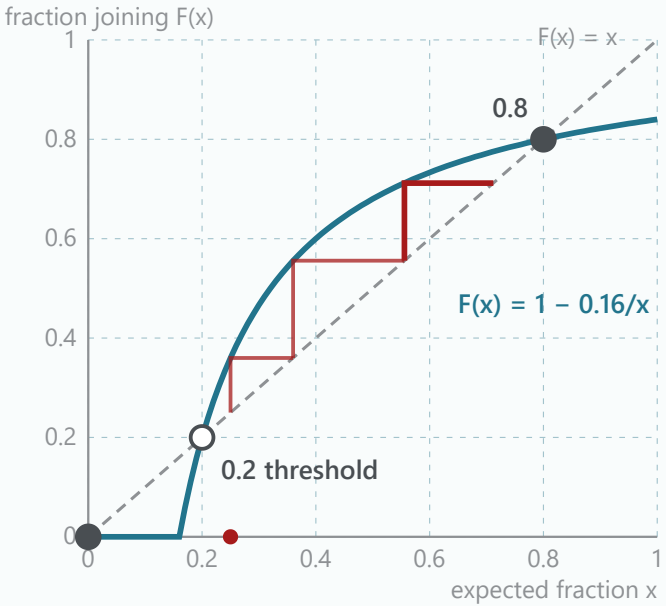
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	
1	0.36	
2	0.556	
3		
4		
5		
6		

Adoption beat expectations; with the new expectation 0.36, those with $\theta \geq 0.444$ join → $x_2 =$ **0.556**.

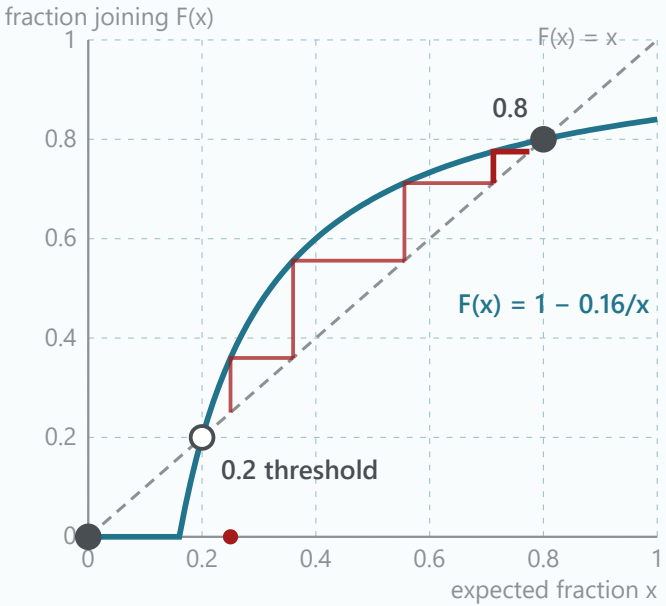
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	
1	0.36	
2	0.556	
3	0.712	
4		
5		
6		

The feedback continues: $x_3 = 1 - 0.16/0.556 =$ **0.712**.

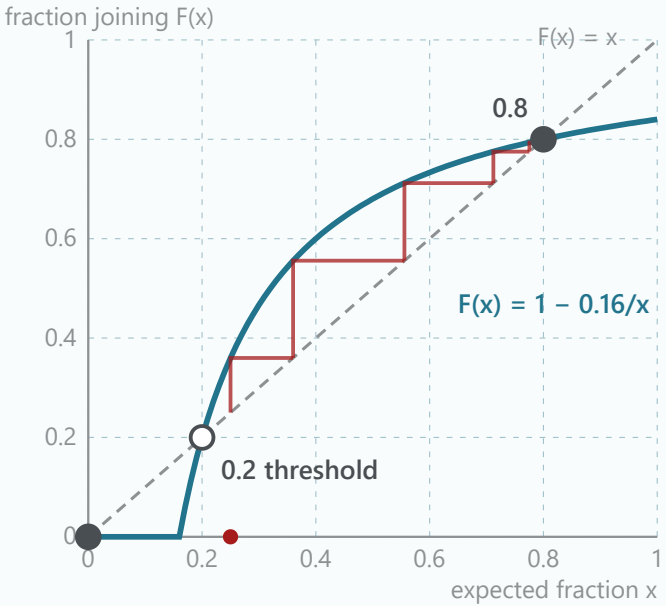
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	
1	0.36	
2	0.556	
3	0.712	
4	0.775	
5		
6		

x_4 = **0.775**; the steps shrink as the path approaches the stable equilibrium.

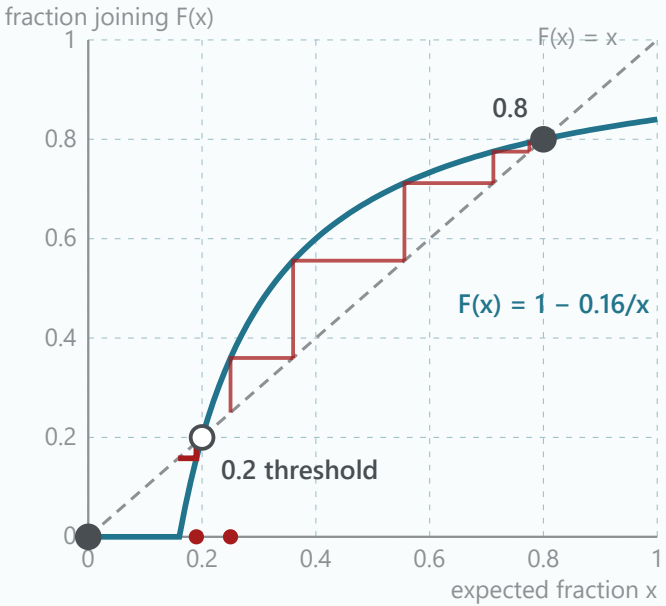
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	
1	0.36	
2	0.556	
3	0.712	
4	0.775	
5	0.794	
6	0.798	

$x_6 \approx 0.798 \to 0.8$: the market has **tipped** in the platform's favor (800 users).

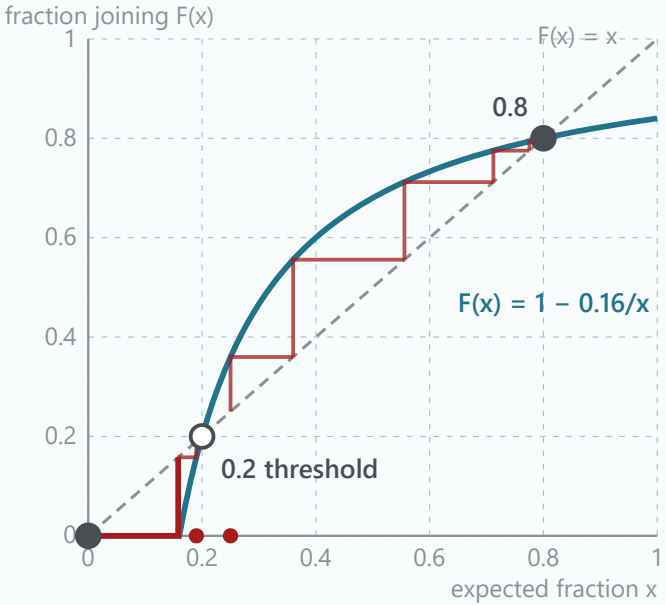
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	0.19
1	0.36	0.158
2	0.556	
3	0.712	
4	0.775	
5	0.794	
6	0.798	

Start just below the threshold: $x_0 = 0.19 \rightarrow x_1 = 1 - 0.16/0.19 =$ **0.158**. Adoption fell short of expectations.

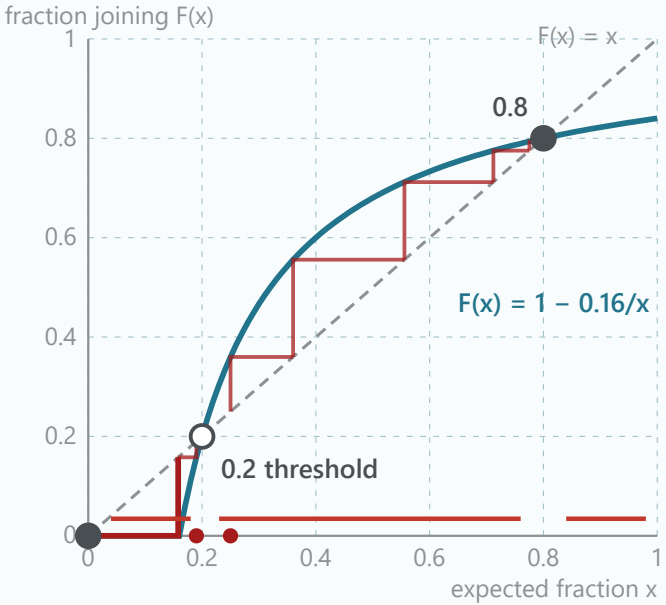
Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	0.19
1	0.36	0.158
2	0.556	0
3	0.712	
4	0.775	
5	0.794	
6	0.798	

$x_1 = 0.158 \le p = 0.16 \rightarrow x_2 =$ **0**: nobody joins and the platform collapses.

Step by Step: Critical Mass and Tipping



t	$x_0 = 0.25$	$x_0 = 0.19$
0	0.25	0.19
1	0.36	0.158
2	0.556	0
3	0.712	
4	0.775	
5	0.794	
6	0.798	

Result: 0 and 0.8 are stable, 0.2 is unstable. **Critical mass is 0.2 (200 users)**: above it the market goes to 0.8, below it to 0.

Reaching Critical Mass: Design Lessons

p	equilibria	threshold
0.09	0, 0.1, 0.9	0.1
0.16	0, 0.2, 0.8	0.2
0.21	0, 0.3, 0.7	0.3
0.25	0 and 0.5 (tangent)	0.5
0.3	only 0	—

A lower fee means a lower threshold. For $x^2 - x + p = 0$ to have real roots, the discriminant $1 - 4p \geq 0$, i.e. $p \leq 0.25$; if $p > 0.25$ the market never forms.

Launch subsidies

Free periods, bonuses: p falls, and so does the critical mass.

Seeding

The platform supplies the first content, sellers or drivers itself.

Managing expectations

An “everyone's here” perception lifts x_0 above the threshold.

Standalone value

Start with a tool that is useful without the network; build the network later.

Example: WhatsApp's value grew rapidly as users joined; Friendster collapsed through the same loop in reverse once users started leaving.

Principles of Two-Sided Pricing

- Besides the overall price **level**, the platform chooses the price **structure**: which side pays, and how much? (Rochet and Tirole, 2003)
- The side that adds more value to the other side, or is more price-sensitive, gets a lower price — even a **subsidy** (a negative price).
- The loss is recovered from fees on the other side, which the subsidized side attracts.

Search / social media
Users are free; advertisers pay.

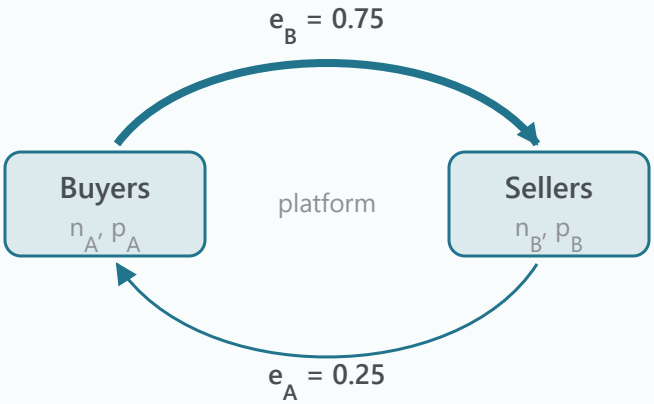
Credit cards
Rewards for cardholders, fees from merchants.

Marketplaces
Coupons for buyers, commissions from sellers.

Revenue models
Commissions, subscriptions, ads, per-transaction fees.

In a simple linear model (Armstrong, 2006) the rule is: **p_A = markup − the value one A participant adds to side B**. If the second term is large, the price turns negative. Markup: the gap between cost and the price the platform would charge a participant without network effects.

A Numerical Example: A Marketplace Model



- n: participants (thousands), p: monthly fee (TL). Serving one more participant costs the platform nothing (marginal cost 0).
- Buyers are very valuable to sellers ($e_B = 0.75$); seller variety matters less to buyers ($e_A = 0.25$).

$$n_A = 20 + 0.25 \cdot n_B - p_A$$
$$n_B = 80 + 0.75 \cdot n_A - p_B$$
$$\Pi = p_A \cdot n_A + p_B \cdot n_B$$

Question: which (p_A, p_B) maximizes profit? Which side is subsidized?

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π

Choosing participation instead of prices simplifies the algebra: each (n_A, n_B) pair corresponds to exactly one price pair.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$

2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π

Substitute into profit. The $n_A n_B$ term shows that the platform can capture the value the two sides add to each other.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$
2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$
3. $\partial\Pi/\partial n_A = 20 + n_B - 2n_A = 0$, $\partial\Pi/\partial n_B = 80 + n_A - 2n_B = 0$

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π

Hold n_B fixed and differentiate Π with respect to n_A (a partial derivative, written ∂); at the optimum both derivatives are 0: two linear equations.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$
2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$
3. $\partial\Pi/\partial n_A = 20 + n_B - 2n_A = 0$, $\partial\Pi/\partial n_B = 80 + n_A - 2n_B = 0$
4. $n_A = 40$, $n_B = 60 \rightarrow p_A = 20 + 15 - 40 = \mathbf{-5}$, $p_B = 80 + 30 - 60 = \mathbf{50}$

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π
Optimum	0.25 / 0.75	−5	50	40	60	**2800**

Solution: buyers **are paid 5 TL** ($p_A = -5$), sellers pay 50 TL. Profit $\Pi = -200 + 3000 = \mathbf{2800}$.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$
2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$
3. $\partial\Pi/\partial n_A = 20 + n_B - 2n_A = 0$, $\partial\Pi/\partial n_B = 80 + n_A - 2n_B = 0$
4. $n_A = 40$, $n_B = 60 \rightarrow p_A = 20 + 15 - 40 = \mathbf{-5}$, $p_B = 80 + 30 - 60 = \mathbf{50}$

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π
Optimum	0.25 / 0.75	−5	50	40	60	**2800**
Free for buyers	0.25 / 0.75	0	47.5	34.6	58.5	**2776.9**

Compare: if buyers are free ($p_A = 0$), the best $p_B \approx 47.5$; fewer buyers, profit 2776.9 < 2800.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$
2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$
3. $\partial\Pi/\partial n_A = 20 + n_B - 2n_A = 0$, $\partial\Pi/\partial n_B = 80 + n_A - 2n_B = 0$
4. $n_A = 40$, $n_B = 60 \rightarrow p_A = 20 + 15 - 40 = \mathbf{-5}$, $p_B = 80 + 30 - 60 = \mathbf{50}$

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π
Optimum	0.25 / 0.75	−5	50	40	60	**2800**
Free for buyers	0.25 / 0.75	0	47.5	34.6	58.5	**2776.9**
Buyers pay 10 TL	0.25 / 0.75	10	42.5	23.8	55.4	**2592.3**

Charging buyers 10 TL cuts them to 23.8 thousand, and sellers shrink too: profit 2592.3.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$
2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$
3. $\partial\Pi/\partial n_A = 20 + n_B - 2n_A = 0$, $\partial\Pi/\partial n_B = 80 + n_A - 2n_B = 0$
4. $n_A = 40$, $n_B = 60 \rightarrow p_A = 20 + 15 - 40 = \mathbf{-5}$, $p_B = 80 + 30 - 60 = \mathbf{50}$
5. Rule: $p_A = n_A - e_B \cdot n_B = 40 - 0.75 \cdot 60 = -5$ (markup n_A – the value a buyer adds for sellers)

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π
Optimum	0.25 / 0.75	−5	50	40	60	**2800**
Free for buyers	0.25 / 0.75	0	47.5	34.6	58.5	**2776.9**
Buyers pay 10 TL	0.25 / 0.75	10	42.5	23.8	55.4	**2592.3**

Why? One extra buyer adds 0.75·60 = 45 TL of value for sellers, which the platform recovers through p_B. 45 > 40 (the markup) → subsidy.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Step by Step: Which Side Gets Subsidized?

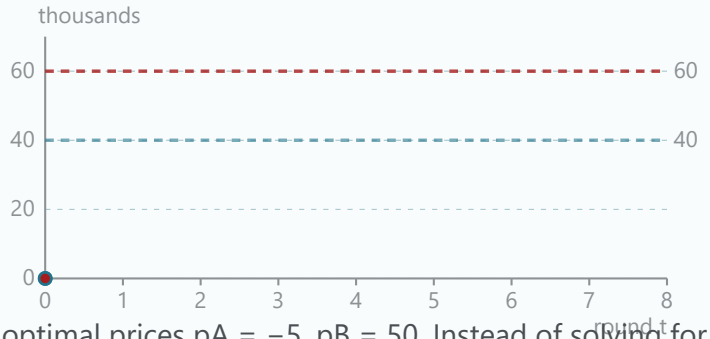
1. Make participation the decision variable: $p_A = 20 + 0.25\, n_B - n_A$, $p_B = 80 + 0.75\, n_A - n_B$
2. $\Pi = 20\, n_A + 80\, n_B + (0.25 + 0.75)\, n_A n_B - n_A^2 - n_B^2$
3. $\partial\Pi/\partial n_A = 20 + n_B - 2n_A = 0$, $\partial\Pi/\partial n_B = 80 + n_A - 2n_B = 0$
4. $n_A = 40$, $n_B = 60 \rightarrow p_A = 20 + 15 - 40 =$ **-5**, $p_B = 80 + 30 - 60 =$ **50**
5. Rule: $p_A = n_A - e_B \cdot n_B = 40 - 0.75 \cdot 60 = -5$ (markup n_A − the value a buyer adds for sellers)

Scenario	e_A / e_B	p_A	p_B	n_A	n_B	Π
Optimum	0.25 / 0.75	−5	50	40	60	**2800**
Free for buyers	0.25 / 0.75	0	47.5	34.6	58.5	**2776.9**
Buyers pay 10 TL	0.25 / 0.75	10	42.5	23.8	55.4	**2592.3**
Effects reversed	0.75 / 0.25	25	30	40	60	**2800**

Counterfactual: with the cross effects reversed (buyers adding little value for sellers), $p_A = 25$ and $p_B = 30$; the subsidy disappears.

A script scanned the (p_A, p_B) grid in 0.5 TL steps: the profit peaks at (−5, 50).

Code Trace: `participation(pA, pB)`



Call with the optimal prices pA = −5, pB = 50. Instead of solving for participation “by hand”, let the sides respond to each other.

Step 1 / 14

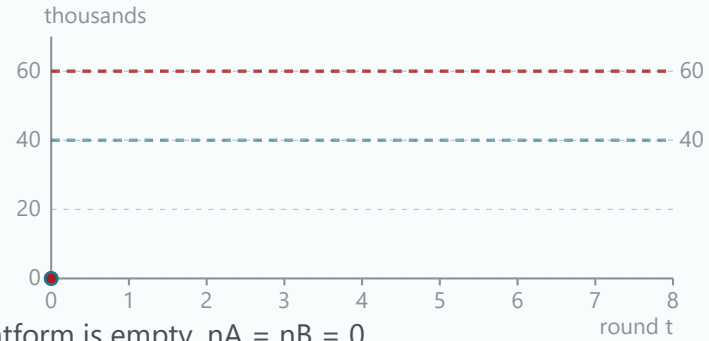
```
▶ def participation(pA, pB, rounds=8):
2     nA, nB = 0, 0
3     for t in range(rounds):
4         nA = 20 + 0.25 * nB - pA    # buyers
5         nB = 80 + 0.75 * nA - pB    # sellers
6     return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `rounds = 8`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 0

nB = 0

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Start: the platform is empty, nA = nB = 0.

Step 2 / 14

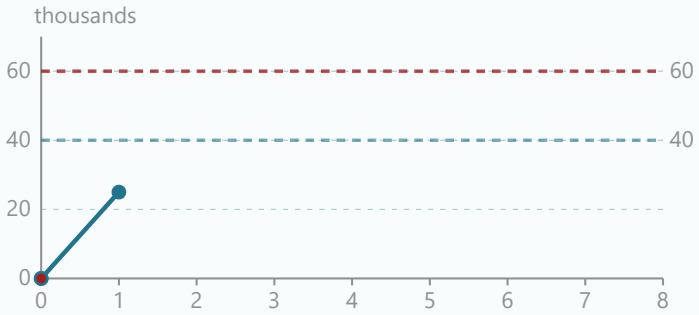
```
1 def participation(pA, pB, rounds=8):
▶   nA, nB = 0, 0
3   for t in range(rounds):
4     nA = 20 + 0.25 * nB - pA   # buyers
5     nB = 80 + 0.75 * nA - pB   # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `nA = 0` `nB = 0`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 25

nB = 0

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Round 1: buyers respond to the current number of sellers: $nA = 20 + 0.25 \cdot 0 + 5 = \mathbf{25}$.

Step 3 / 14

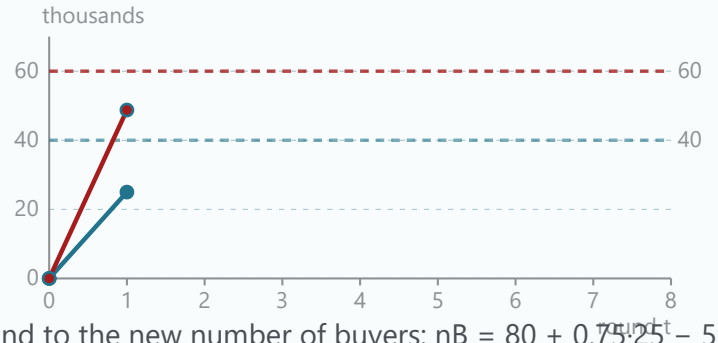
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA    # buyers
5     nB = 80 + 0.75 * nA - pB    # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 0` `nA = 25.0` `nB = 0.0`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 25

nB = 48.8

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Sellers respond to the new number of buyers: nB = 80 + 0.75·25 − 50 = **48.8**. The buyer subsidy drew in sellers.

Step 4 / 14

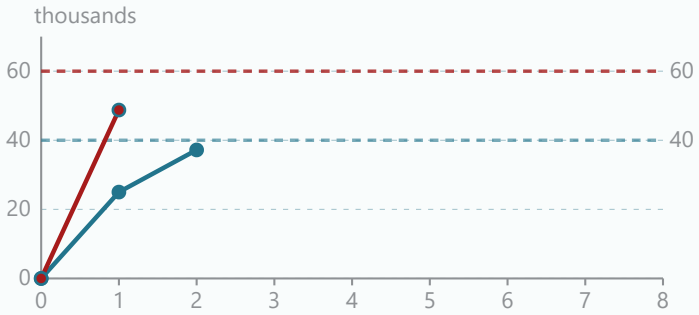
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
4     nA = 20 + 0.25 * nB - pA    # buyers
▶     nB = 80 + 0.75 * nA - pB    # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 0` `nA = 25.0` `nB = 48.75`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 37.2

nB = 48.8

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Round 2: buyers respond to the current number of sellers: $nA = 20 + 0.25 \cdot 48.8 + 5 = \mathbf{37.2}$.

Step 5 / 14

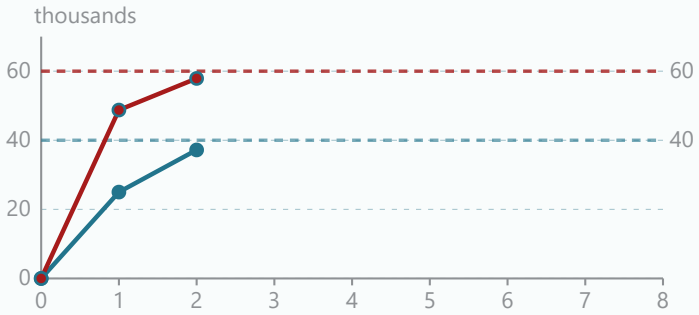
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA    # buyers
5     nB = 80 + 0.75 * nA - pB    # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 1` `nA = 37.19` `nB = 48.75`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 37.2

nB = 57.9

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Sellers respond to the new number of buyers: nB = 80 + 0.75·37.2 − 50 = **57.9**. The cross effect grows both sides together.

Step 6 / 14

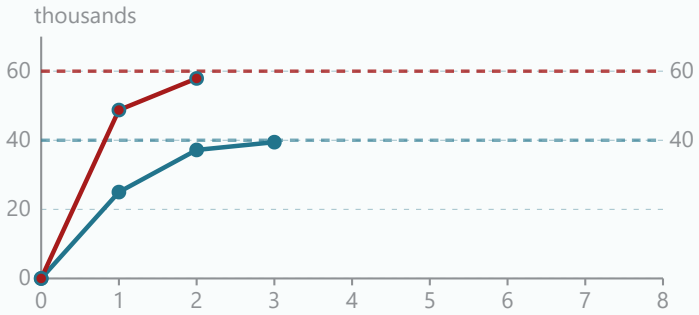
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
4     nA = 20 + 0.25 * nB - pA    # buyers
▶     nB = 80 + 0.75 * nA - pB    # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 1` `nA = 37.19` `nB = 57.89`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 39.5

nB = 57.9

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Round 3: buyers respond to the current number of sellers: $nA = 20 + 0.25 \cdot 57.9 + 5 = \mathbf{39.5}$.

Step 7 / 14

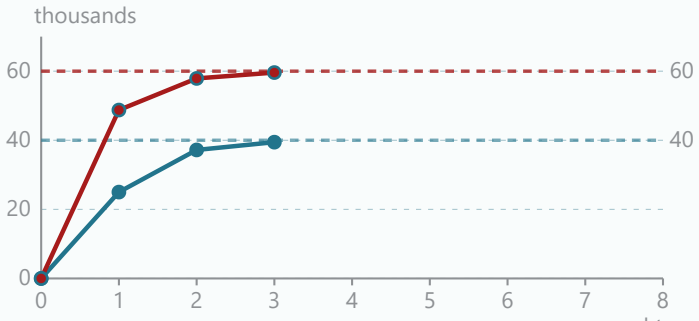
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA    # buyers
5     nB = 80 + 0.75 * nA - pB    # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 2` `nA = 39.47` `nB = 57.89`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 39.5

nB = 59.6

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Sellers respond to the new number of buyers: nB = 80 + 0.75·39.5 − 50 = **59.6**. The cross effect grows both sides together.

Step 8 / 14

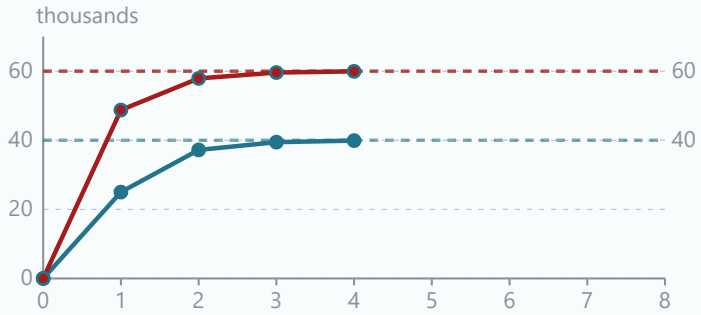
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
4     nA = 20 + 0.25 * nB - pA    # buyers
▶     nB = 80 + 0.75 * nA - pB    # sellers
6   return nA, nB
```

VARIABLES

`pA = −5` `pB = 50` `t = 2` `nA = 39.47` `nB = 59.6`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 39.9

nB = 59.9

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Round 4: nA = 39.9, nB = 59.9. The change shrinks by a factor of ≈ 0.25·0.75 ≈ 0.19 each round.

Step 9 / 14

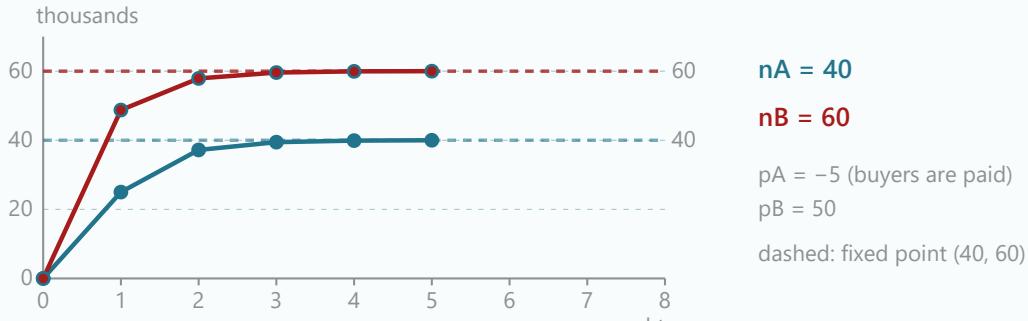
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA   # buyers
▶     nB = 80 + 0.75 * nA - pB   # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 3` `nA = 39.9` `nB = 59.93`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



Round 5: nA = 40, nB = 60. The change shrinks by a factor of ≈ 0.25·0.75 ≈ 0.19 each round.

Step 10 / 14

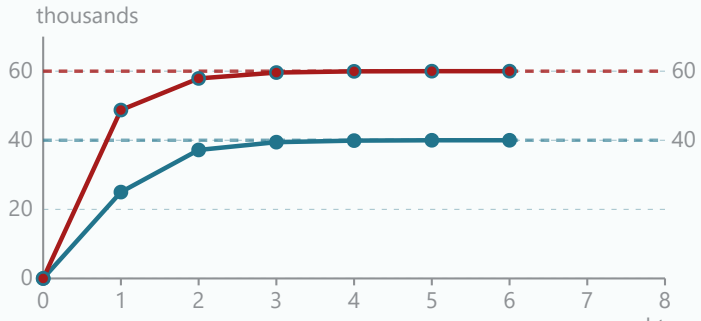
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA   # buyers
▶     nB = 80 + 0.75 * nA - pB   # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 4` `nA = 39.98` `nB = 59.99`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



Round 6: nA = 40, nB = 60. The change shrinks by a factor of ≈ 0.25·0.75 ≈ 0.19 each round.

Step 11 / 14

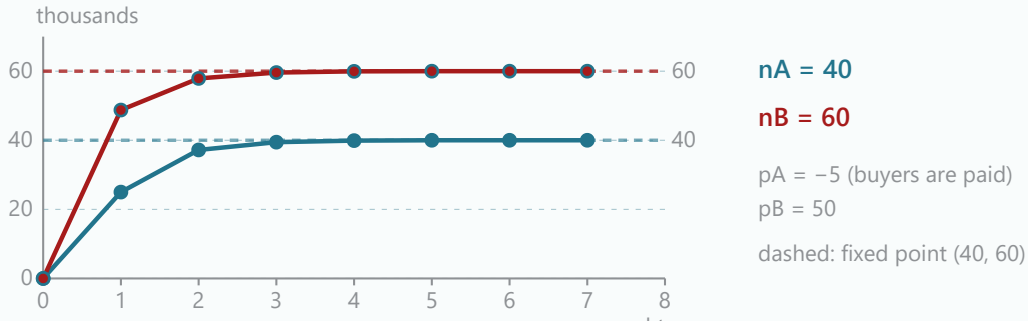
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA   # buyers
▶     nB = 80 + 0.75 * nA - pB   # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 5` `nA = 40.0` `nB = 60.0`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



Round 7: nA = 40, nB = 60. The change shrinks by a factor of ≈ 0.25·0.75 ≈ 0.19 each round.

Step 12 / 14

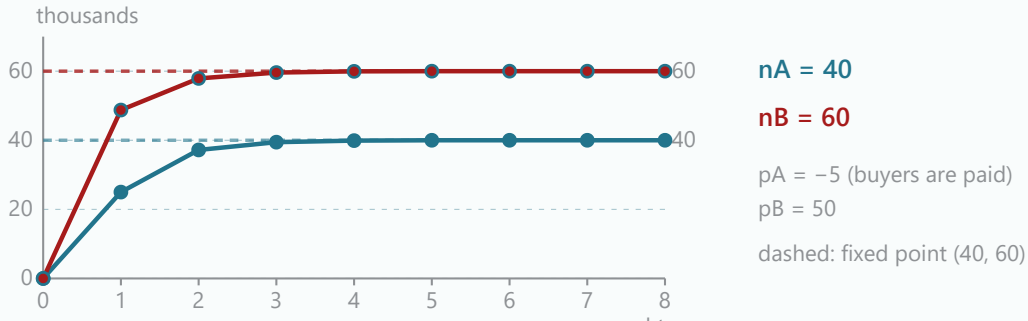
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA   # buyers
▶     nB = 80 + 0.75 * nA - pB   # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 6` `nA = 40.0` `nB = 60.0`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



Round 8: nA = 40, nB = 60. The change shrinks by a factor of ≈ 0.25·0.75 ≈ 0.19 each round.

Step 13 / 14

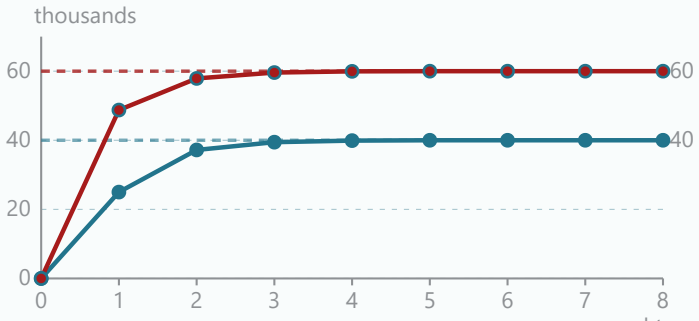
```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
▶     nA = 20 + 0.25 * nB - pA   # buyers
▶     nB = 80 + 0.75 * nA - pB   # sellers
6   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `t = 7` `nA = 40.0` `nB = 60.0`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

Code Trace: `participation(pA, pB)`



nA = 40

nB = 60

pA = −5 (buyers are paid)
pB = 50

dashed: fixed point (40, 60)

Fixed point (40, 60). Profit = −5·40 + 50·60 = **2,800**. This is the participation equilibrium where each side's expectations confirm the other's.

Step 14 / 14

```
1 def participation(pA, pB, rounds=8):
2   nA, nB = 0, 0
3   for t in range(rounds):
4     nA = 20 + 0.25 * nB - pA   # buyers
5     nB = 80 + 0.75 * nA - pB   # sellers
▶   return nA, nB
```

VARIABLES

`pA = -5` `pB = 50` `nA = 40.0` `nB = 60.0`

Convergence condition: the product of the cross effects $e_A \cdot e_B = 0.1875 < 1$. If it exceeded 1, participation would explode or collapse: the network effect is too strong (tipping).

A Richer Model: Two-Sided Platform Pricing

The same model, two more parameters

Model: $n_A = base_A - s_A \cdot p_A + e_A \cdot n_B$, $n_B = base_B - s_B \cdot p_B + e_B \cdot n_A$.

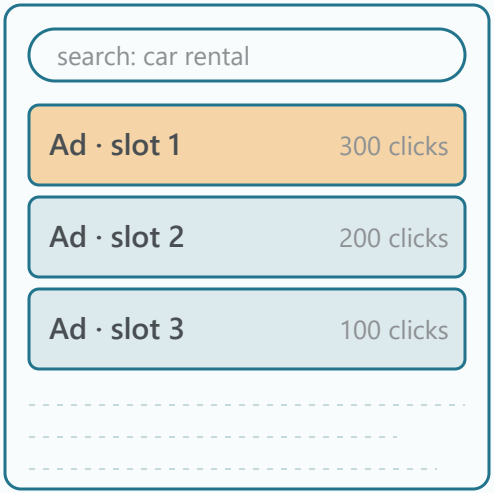
- base: users who join at a zero fee when the other side is empty.
- s: price sensitivity (marketplace: $s_A = 11$, $s_B = 5$); $s = 1$ in the numerical model.
- Participation is capped at 0–100 users; fees are integers from −5 to 10.
- Profit $\Pi = p_A \cdot n_A + p_B \cdot n_B$; participation is again the fixed point of mutual responses.

Level	e_A / e_B	Most profitable (p_A, p_B)	n_A / n_B	Π
1	0.10 / 0.30	(2, 4)	25.3 / 22.6	140.8
2	0.15 / 0.60	(0, 6)	50.3 / 35.2	211.0
3	0.20 / 0.85	**(−2, 9)**	78.3 / 56.6	352.5

Marketplace scenario (base 45 / 35). At level 3, fees of (3, 3) earn only 167.0: subsidizing A and charging B a high fee more than doubles the profit.

The more side B values A (the larger e_B), the lower A's fee, until it turns negative: the model's version of the rule p_A = markup − the value A adds to side B.

Sponsored Search Auctions



- Every search query triggers an auction among the advertisers bidding on that keyword.
- m slots with click counts $\alpha_1 > \alpha_2 > \ldots$ from top to bottom (here 300, 200, 100 per 1000 impressions).
- Advertiser i values a click at v_i, bids b_i per click, and in slot j pays $\mathbf{p_j}$ **per click**.
- The utility of the advertiser in slot j: $\alpha_j \cdot (v_i - p_j)$.

Example: A, B, C, D value a click at 10, 8, 5, 2 TL (and bid that). Three slots, four advertisers.

Two Payment Rules: GSP and VCG

GSP (generalized second price)

Rank by bid ($b_{(k)}$: the k-th highest bid). The advertiser in slot j pays **the next bid down** per click:

$$p_j = b_{(j+1)}$$

With one slot it equals Vickrey; with several slots truthfulness is lost.

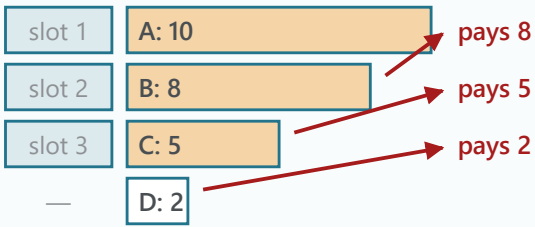
VCG

Same ranking. Each advertiser pays **the loss its presence imposes on others**:

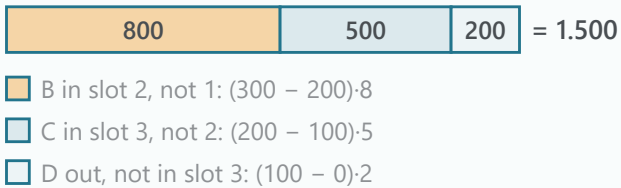
$$\alpha_j \cdot p_j = \Sigma_{k \geq j} (\alpha_k - \alpha_{k+1}) \cdot b_{(k+1)}$$

Truthful bidding is a dominant strategy.

GSP: per click, the next bid down



VCG: A's total payment = the losses it causes



So under VCG, the advertiser in slot j pays the total click value lost by those below as each slides down one slot. $\alpha_j \cdot p_j$ is the total payment, p_j the per-click payment; $\alpha_{m+1} = 0$.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)				
2	200	B (b = 8)				
3	100	C (b = 5)				
—	0	D (b = 2)				
		Revenue				

Ranked by bid: A → slot 1, B → slot 2, C → slot 3; D is left out.

Both rules get the same bids (= values); only the payments differ.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400		
2	200	B (b = 8)				
3	100	C (b = 5)				
—	0	D (b = 2)	—	0		
		Revenue				

GSP: A pays B's bid, the one below: 8 TL/click × 300 = **2400**.

Under GSP each slot pays the next bid down, per click.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400		
2	200	B (b = 8)	5	1000		
3	100	C (b = 5)				
—	0	D (b = 2)	—	0		
		Revenue				

GSP: B pays C's bid: 5 × 200 = **1000**.

The second slot.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400		
2	200	B (b = 8)	5	1000		
3	100	C (b = 5)	2	200		
—	0	D (b = 2)	—	0		
		Revenue		**3600**		

GSP: C pays D's bid: 2 × 100 = **200**. GSP revenue is **3600**.

GSP done.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400		
2	200	B (b = 8)	5	1000		
3	100	C (b = 5)	2	200	200	2
—	0	D (b = 2)	—	0	0	—
		Revenue		**3600**		

VCG, C: without C, D moves up to slot 3: welfare below is 200; with C it is 0 → C pays **200**.

VCG payment = others' welfare “if it were absent” − others' welfare with it present. Those above are unaffected; only those below count.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400		
2	200	B (b = 8)	5	1000	700	3.5
3	100	C (b = 5)	2	200	200	2
—	0	D (b = 2)	—	0	0	—
		Revenue		**3600**		

VCG, B: without B, C takes slot 2 (200·5) and D slot 3 (100·2) → 1200; with B, C gets 100·5 = 500 → **700**.

Removing B would move C up one slot and bring D into slot 3.

Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400	1500	5
2	200	B (b = 8)	5	1000	700	3.5
3	100	C (b = 5)	2	200	200	2
—	0	D (b = 2)	—	0	0	—
		Revenue		**3600**	**2400**	

VCG, A: without A, 300·8 + 200·5 + 100·2 = 3600; with A, 200·8 + 100·5 = 2100 → **1500**.

The formula gives the same: (300−200)·8 + (200−100)·5 + 100·2 = 1500.

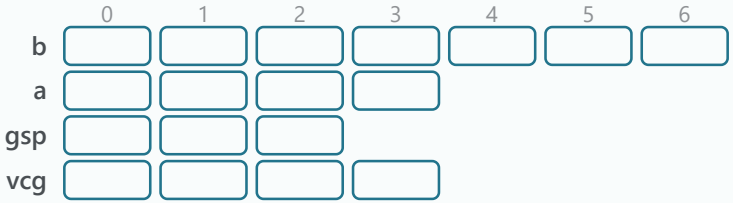
Step by Step: GSP and VCG Payments from the Same Bids

Slot	Clicks	Advertiser	GSP /click	GSP total	VCG total	VCG /click
1	300	A (b = 10)	8	2400	1500	5
2	200	B (b = 8)	5	1000	700	3.5
3	100	C (b = 5)	2	200	200	2
—	0	D (b = 2)	—	0	0	—
		Revenue		**3600**	**2400**	

With the same bids, GSP charges at least as much as VCG in every slot: 8 ≥ 5, 5 ≥ 3.5, 2 ≥ 2 TL/click.

Revenue: GSP **3600**, VCG **2400**. But the comparison is unfair: under GSP the bids change.

Code Trace: `gsp_vcg(bids, ctr)`



b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

m = 3 slots.

Step 1 / 14

```
1  def gsp_vcg(bids, ctr):
▶     m = len(ctr)
3     b = sorted(bids, reverse=True) + [0]*m
4     a = ctr + [0]      # dummy slot: 0 clicks
5     gsp, vcg = [0]*m, [0]*(m + 1)
6     for j in range(m):
7       gsp[j] = a[j] * b[j+1]   # next bid
8     for j in range(m-1, -1, -1):  # bottom up
9       extra = (a[j] - a[j+1]) * b[j+1]
10      vcg[j] = vcg[j+1] + extra
11    return gsp, vcg[:m]
```

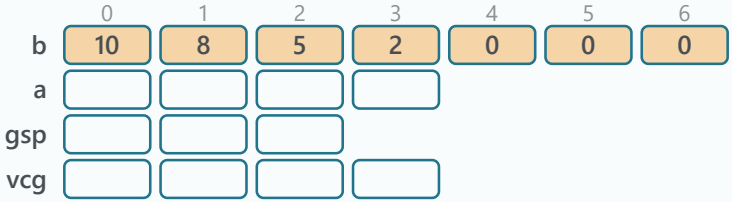
VARIABLES

`bids = [5, 10, 2, 8]`

`ctr = [300, 200, 100]` `m = 3`

SPONSORED SEARCH

Code Trace: `gsp_vcg(bids, ctr)`



b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

Sort bids from high to low and append m zeros so that a “next bid down” always exists.

Step 2 / 14

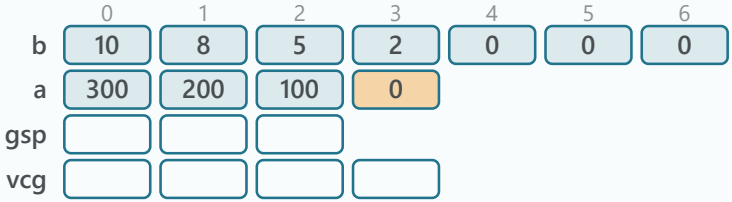
```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
▶    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3`

`b = [10, 8, 5, 2, 0, 0, 0]`

Code Trace: `gsp_vcg(bids, ctr)`



b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

Append a dummy (m+1)-th slot with 0 clicks to a: $\alpha_{m+1} = 0$.

Step 3 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
▶    a = ctr + [0]      # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

Code Trace: `gsp_vcg(bids, ctr)`

	0	1	2	3	4	5	6	
b	10	8	5	2	0	0	0	b: sorted bids (TL/click)
a	300	200	100	0				a: slot clicks
gsp	0	0	0					gsp[j]: total GSP payment
vcg	0	0	0	0				vcg[j]: total VCG payment

Zero the result arrays; vcg's extra last element is the recursion's starting value.

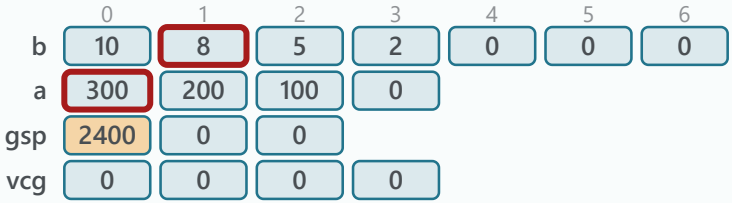
Step 4 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
▶    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]` `a = [300, 200, 100, 0]` `gsp = [0, 0, 0]` `vcg = [0, 0, 0, 0]`

Code Trace: `gsp_vcg(bids, ctr)`



j = 0

b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

gsp[0] = a[0]·b[1] = 300·8 = **2400**: slot 1 pays the next bid down, per click.

Step 5 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
▶      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

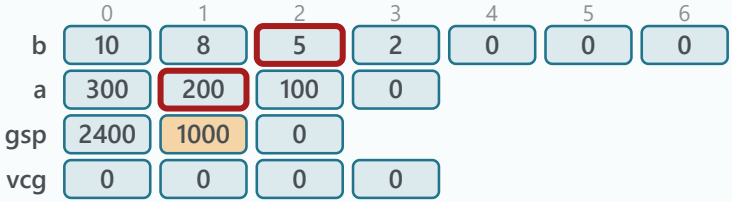
`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

`gsp = [2400, 0, 0]`

`vcg = [0, 0, 0, 0]` `j = 0`

Code Trace: `gsp_vcg(bids, ctr)`



j = 1

b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

gsp[1] = a[1]·b[2] = 200·5 = **1000**: slot 2 pays the next bid down, per click.

Step 6 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]      # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
▶      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

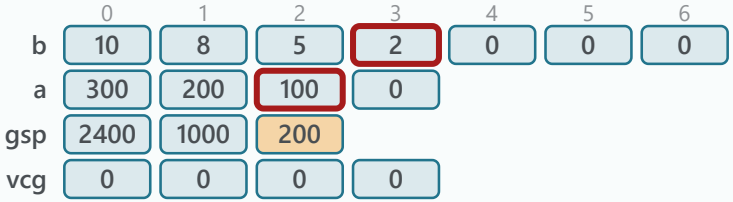
`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 0]`

`vcg = [0, 0, 0, 0]` `j = 1`

Code Trace: `gsp_vcg(bids, ctr)`



b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

j = 2

gsp[2] = a[2]·b[3] = 100·2 = **200**: slot 3 pays the next bid down, per click.

Step 7 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]      # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
▶      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

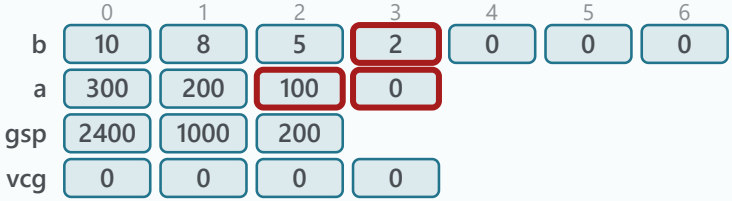
`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [0, 0, 0, 0]` `j = 2`

Code Trace: `gsp_vcg(bids, ctr)`



extra = (100 − 0)·2 = **200**: the click value lost by the advertiser below, stuck in slot 4 instead of 3.

Step 8 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]      # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
▶      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [0, 0, 0, 0]` `j = 2`

`extra = 200`

Code Trace: `gsp_vcg(bids, ctr)`

	0	1	2	3	4	5	6
b	10	8	5	2	0	0	0
a	300	200	100	0			
gsp	2400	1000	200				
vcg	0	0	200	0			

j = 2

b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

vcg[2] = 0 + 200 = **200**: losses imposed further down carry upward too.

Step 9 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
▶      vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

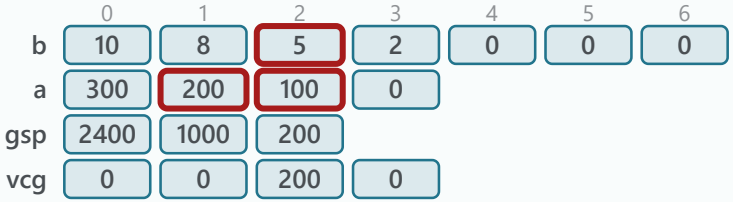
`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [0, 0, 200, 0]` `j = 2`

`extra = 200`

Code Trace: `gsp_vcg(bids, ctr)`



extra = (200 − 100)·5 = **500**: the click value lost by the advertiser below, stuck in slot 3 instead of 2.

Step 10 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]      # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
▶      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [0, 0, 200, 0]` `j = 1`

`extra = 500`

Code Trace: `gsp_vcg(bids, ctr)`

	0	1	2	3	4	5	6
b	10	8	5	2	0	0	0
a	300	200	100	0			
gsp	2400	1000	200				
vcg	0	700	200	0			

j = 1

b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

vcg[1] = 200 + 500 = **700**: losses imposed further down carry upward too.

Step 11 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
▶      vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

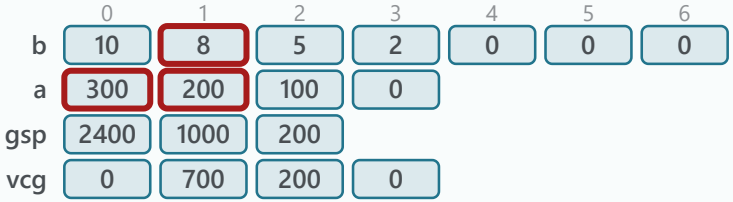
`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [0, 700, 200, 0]` `j = 1`

`extra = 500`

Code Trace: `gsp_vcg(bids, ctr)`



j = 0

b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

extra = (300 − 200)·8 = **800**: the click value lost by the advertiser below, stuck in slot 2 instead of 1.

Step 12 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]      # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
▶      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

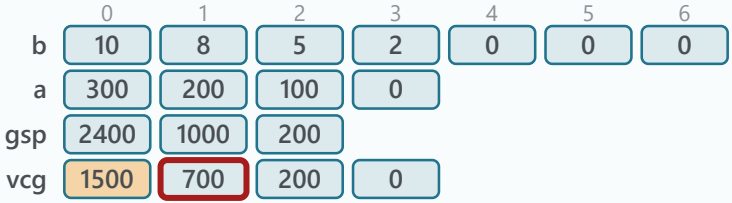
`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [0, 700, 200, 0]` `j = 0`

`extra = 800`

Code Trace: `gsp_vcg(bids, ctr)`



b: sorted bids (TL/click)

a: slot clicks

gsp[j]: total GSP payment

vcg[j]: total VCG payment

j = 0

vcg[0] = 700 + 800 = **1500**: losses imposed further down carry upward too.


```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
▶      vcg[j] = vcg[j+1] + extra
11   return gsp, vcg[:m]
```

VARIABLES

`m = 3` `b = [10, 8, 5, 2, 0, 0, 0]`

`a = [300, 200, 100, 0]`

`gsp = [2400, 1000, 200]`

`vcg = [1500, 700, 200, 0]` `j = 0`

`extra = 800`

Code Trace: `gsp_vcg(bids, ctr)`

	0	1	2	3	4	5	6	
b	10	8	5	2	0	0	0	b: sorted bids (TL/click)
a	300	200	100	0				a: slot clicks
gsp	2400	1000	200					gsp[j]: total GSP payment
vcg	1500	700	200	0				vcg[j]: total VCG payment

Result: GSP [2400, 1000, 200] (total 3600), VCG [1500, 700, 200] (total 2400).

Step 14 / 14

```
1  def gsp_vcg(bids, ctr):
2    m = len(ctr)
3    b = sorted(bids, reverse=True) + [0]*m
4    a = ctr + [0]     # dummy slot: 0 clicks
5    gsp, vcg = [0]*m, [0]*(m + 1)
6    for j in range(m):
7      gsp[j] = a[j] * b[j+1]   # next bid
8    for j in range(m-1, -1, -1):  # bottom up
9      extra = (a[j] - a[j+1]) * b[j+1]
10     vcg[j] = vcg[j+1] + extra
▶    return gsp, vcg[:m]
```

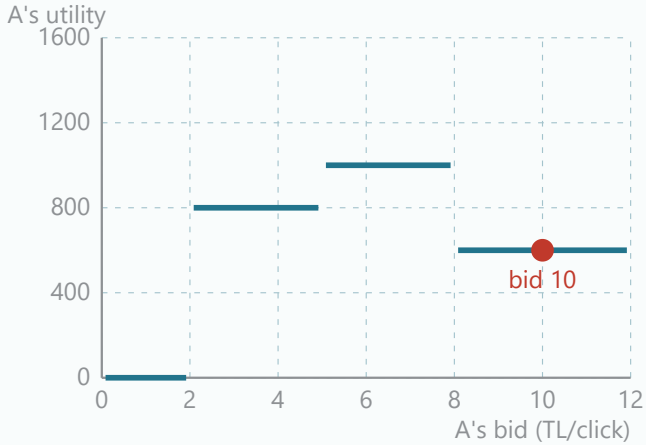
VARIABLES

`gsp = [2400, 1000, 200]`

`vcg = [1500, 700, 200]`

Step by Step: Truthful Bidding Is Not an Equilibrium of GSP

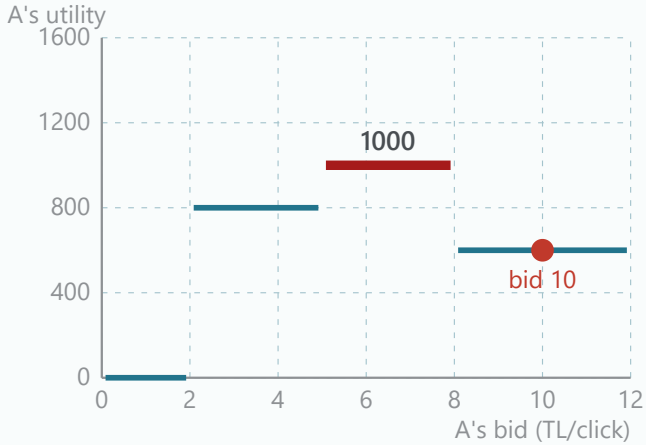
Adv.	v	bid	slot	pays/click	utility
A	10	10	1	8	**600**
B	8	8	2	5	**600**
C	5	5	3	2	**300**
D	2	2	—	—	**0**
Platform revenue					**3600**



Everyone bids their value. A is in slot 1 and pays B's bid (8): utility 300·(10 − 8) = **600**.

Step by Step: Truthful Bidding Is Not an Equilibrium of GSP

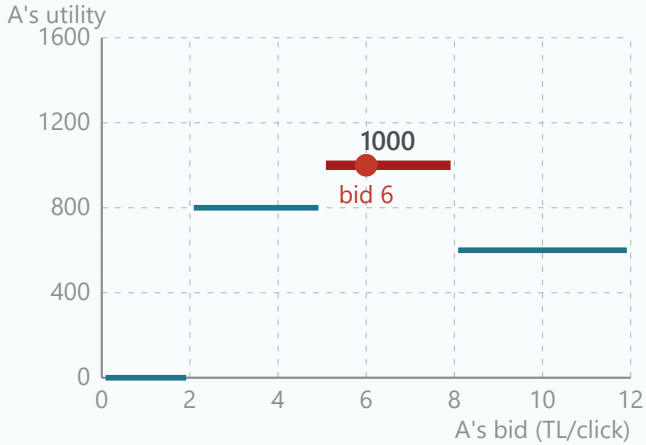
Adv.	v	bid	slot	pays/click	utility
A	10	10	1	8	**600**
B	8	8	2	5	**600**
C	5	5	3	2	**300**
D	2	2	—	—	**0**
Platform revenue					**3600**



A's utility is a step function of its bid: 2–5 gives slot 3 (800), **5–8 gives slot 2 (1000)**, above 8 slot 1 (600).

Step by Step: Truthful Bidding Is Not an Equilibrium of GSP

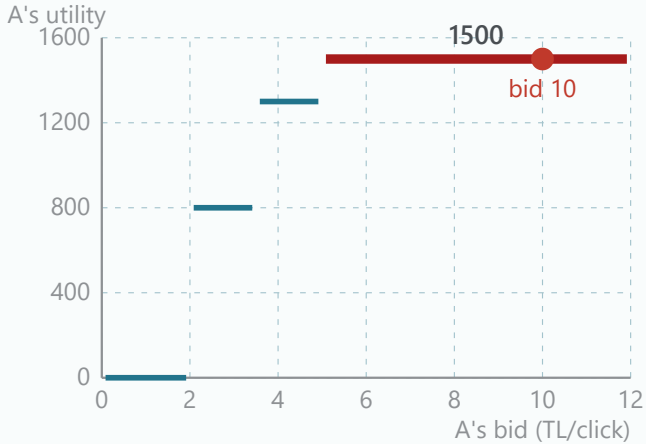
Adv.	v	bid	slot	pays/click	utility
B	8	8	1	6	**600**
A	10	6	2	5	**1000**
C	5	5	3	2	**300**
D	2	2	—	—	**0**
Platform revenue					**3000**



If A bids 6, it drops to slot 2 and pays C's 5: 200·(10 − 5) = **1000 > 600**. Truthful bidding is not a best response → **not an equilibrium**.

Step by Step: Truthful Bidding Is Not an Equilibrium of GSP

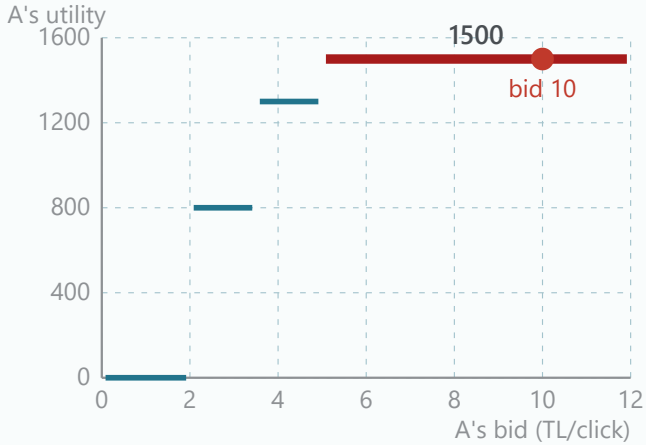
Adv.	v	bid	slot	pays/click	utility
A	10	10	1	5	**1500**
B	8	5	2	3.5	**900**
C	5	3.5	3	2	**300**
D	2	2	—	—	**0**
Platform revenue					**2400**



An equilibrium: B bids 5, C 3.5, D 2. A's best is still slot 1 (1500); a script tried every deviation, and nobody gains.

Step by Step: Truthful Bidding Is Not an Equilibrium of GSP

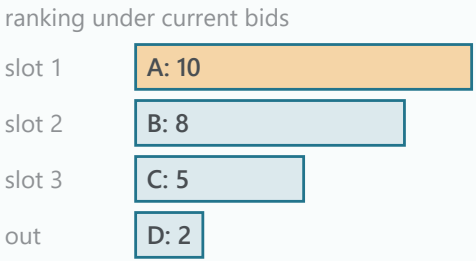
Adv.	v	bid	slot	pays/click	utility
A	10	10	1	5	**1500**
B	8	5	2	3.5	**900**
C	5	3.5	3	2	**300**
D	2	2	—	—	**0**
Platform revenue					**2400**



In this equilibrium the per-click payments 5 / 3.5 / 2 are the **VCG payments**, revenue 2400. It is the lowest-revenue locally envy-free equilibrium of GSP. Locally envy-free: no advertiser would rather take the slot just above at that slot's current per-click price.

SPONSORED SEARCH

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

Truthful bids b = v = [10, 8, 5, 2]. Should A (i = 0) deviate? Candidate bids 9, 6, 4, 3, 1.

Step 1 / 12

```
▶ def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
3   for x in cands:
4     b2 = b[:]; b2[i] = x   # only i changes
5     u = gsp_utility(i, b2, v, ctr)
6     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

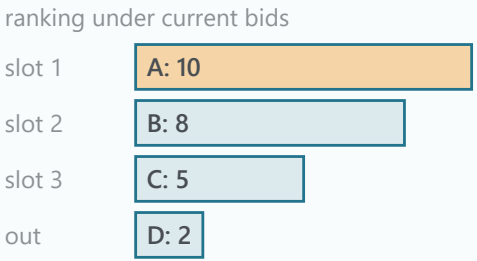
VARIABLES

`i = 0` `b = [10, 8, 5, 2]`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

SPONSORED SEARCH

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 600

Currently A is in slot 1 and pays the next bid down (8): old = 300·(10 − 8) = **600**.

Step 2 / 12

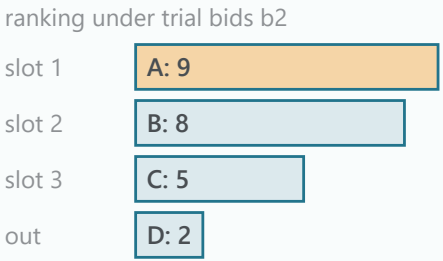
```
1 def profitable_dev(i, b, v, ctr, cands):
▶   old = gsp_utility(i, b, v, ctr)
3   for x in cands:
4     b2 = b[:]; b2[i] = x   # only i changes
5     u = gsp_utility(i, b2, v, ctr)
6     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 8, 5, 2]` `old = 600`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 600

x = 9 → slot 1, u = 600

u > old fails → continue

x = 9: slot 1, u = 600 ≤ 600.

Step 3 / 12

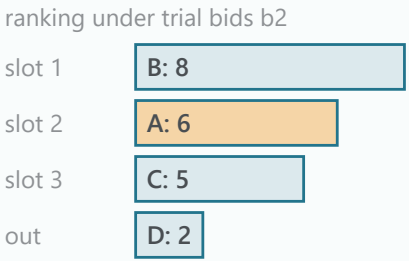
```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
▶   for x in cands:
▶     b2 = b[:]; b2[i] = x   # only i changes
▶     u = gsp_utility(i, b2, v, ctr)
▶     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 8, 5, 2]` `old = 600` `x = 9` `u = 600`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 600

x = 6 → slot 2, u = 1,000

u > old → profitable deviation!

return: (6, 1,000)

x = 6: A drops to slot 2 and pays less: u = **1,000** > 600 → **profitable deviation found**; return (x, u).

Step 4 / 12

```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
▶   for x in cands:
▶     b2 = b[:]; b2[i] = x    # only i changes
▶     u = gsp_utility(i, b2, v, ctr)
▶     if u > old:
▶       return x, u    # profitable
8   return None      # no deviation
```

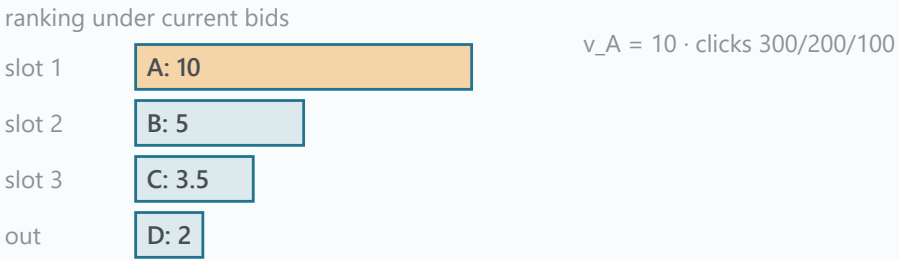
VARIABLES

`i = 0` `b = [10, 8, 5, 2]` `old = 600` `x = 6` `u = 1000`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

SPONSORED SEARCH

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



Now the proposed equilibrium: B bids 5, C 3.5, D 2. Same candidate list for A.

Step 5 / 12

```
▶ def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
3   for x in cands:
4     b2 = b[:]; b2[i] = x   # only i changes
5     u = gsp_utility(i, b2, v, ctr)
6     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

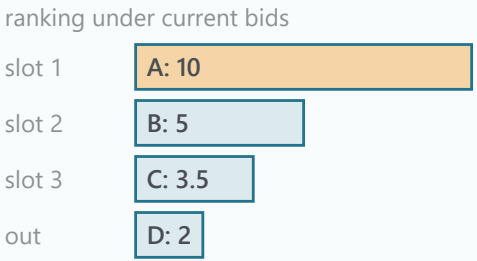
VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

SPONSORED SEARCH

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

Currently A is in slot 1 and pays the next bid down (5): old = 300·(10 − 5) = **1,500**.

Step 6 / 12

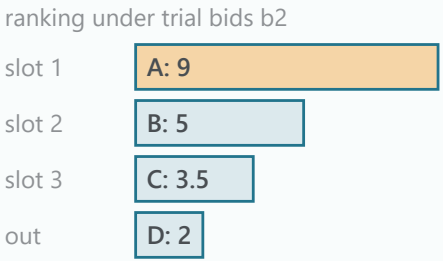
```
1 def profitable_dev(i, b, v, ctr, cands):
▶   old = gsp_utility(i, b, v, ctr)
3   for x in cands:
4     b2 = b[:]; b2[i] = x   # only i changes
5     u = gsp_utility(i, b2, v, ctr)
6     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

x = 9 → slot 1, u = 1,500

u > old fails → continue

x = 9: slot 1, u = 1,500 ≤ 1,500.

Step 7 / 12

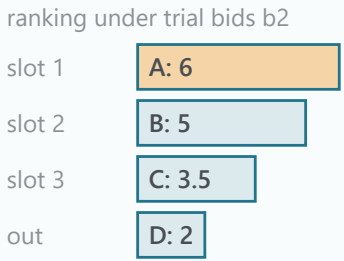
```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
▶   for x in cands:
▶     b2 = b[:]; b2[i] = x   # only i changes
▶     u = gsp_utility(i, b2, v, ctr)
▶     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500` `x = 9` `u = 1500`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

x = 6 → slot 1, u = 1,500

u > old fails → continue

x = 6: slot 1, u = 1,500 ≤ 1,500.

Step 8 / 12

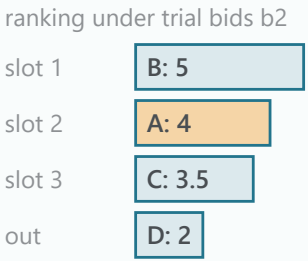
```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
    for x in cands:
      b2 = b[:]; b2[i] = x   # only i changes
      u = gsp_utility(i, b2, v, ctr)
      if u > old:
7       return x, u   # profitable
8   return None      # no deviation
```

VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500` `x = 6` `u = 1500`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

x = 4 → slot 2, u = 1,300

u > old fails → continue

x = 4: slot 2, u = 1,300 ≤ 1,500.

Step 9 / 12

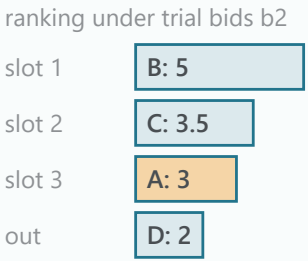
```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
▶   for x in cands:
▶     b2 = b[:]; b2[i] = x   # only i changes
▶     u = gsp_utility(i, b2, v, ctr)
▶     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500` `x = 4` `u = 1300`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

x = 3 → slot 3, u = 800

u > old fails → continue

x = 3: slot 3, u = 800 ≤ 1,500.

Step 10 / 12

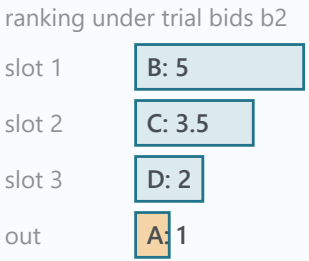
```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
▶   for x in cands:
▶     b2 = b[:]; b2[i] = x   # only i changes
▶     u = gsp_utility(i, b2, v, ctr)
▶     if u > old:
7       return x, u   # profitable
8   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500` `x = 3` `u = 800`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

x = 1 → no slot, u = 0

u > old fails → continue

x = 1: the ad is left out, u = 0 ≤ 1,500.

Step 11 / 12

```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
    for x in cands:
      b2 = b[:]; b2[i] = x   # only i changes
      u = gsp_utility(i, b2, v, ctr)
      if u > old:
7       return x, u   # profitable
8   return None      # no deviation
```

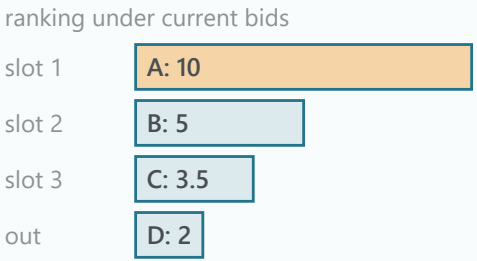
VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500` `x = 1` `u = 0`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

SPONSORED SEARCH

Code Trace: `profitable_dev(i, b, v, ctr, cands)`



v_A = 10 · clicks 300/200/100

old utility = 1,500

return: None

No candidate is profitable → **None**. A script ran the same test for all four advertisers on a bid grid with step 0.25: nobody gains, so this profile is a Nash equilibrium.

Step 12 / 12

```
1 def profitable_dev(i, b, v, ctr, cands):
2   old = gsp_utility(i, b, v, ctr)
3   for x in cands:
4     b2 = b[:]; b2[i] = x   # only i changes
5     u = gsp_utility(i, b2, v, ctr)
6     if u > old:
7       return x, u   # profitable
▶   return None     # no deviation
```

VARIABLES

`i = 0` `b = [10, 5, 3.5, 2]` `old = 1500`

gsp_utility(i, b, v, ctr): sort the bids; if i is in slot j, ctr[j]·(v[i] − the next bid down), and 0 if it gets no slot. Bids are assumed distinct.

GSP and VCG Compared

	GSP	VCG
Is truthful bidding dominant?	No (A bids 6 instead of 10)	Yes
Revenue with the same bids	3600 (≥ VCG in every slot)	2400
Revenue in equilibrium	≥ 2400 in locally envy-free equilibria; the lowest equals VCG	2400 (with truthful bids)
Explaining it	Easy: “the next bid down”	A sum of externalities, harder
In practice	Long the norm in search ads (with quality scores)	Used in Facebook's ad auction (Varian and Harris, 2014)

Quality score

In practice, ranking uses the ad's click quality (its quality score) together with the bid.

Lesson

The name “second price” does not guarantee truthfulness: what holds for one item can break with many slots.

SPONSORED SEARCH

A Given Bid Profile ≠ Equilibrium Bids

GSP outcomes are computed for given bids. 3600 is the revenue when everyone bids truthfully — but truthful bidding is not a dominant strategy in GSP. Click counts 300, 200, 100; values A 10, B 8, C 5, D 2.

Profile	Bids (A, B, C, D)	A's slot and utility	Platform revenue
Truthful	10, 8, 5, 2	slot 1: 300·(10 − 8) = 600	3600
A bids lower	6, 8, 5, 2	slot 2: 200·(10 − 5) = **1000**	3000
Locally envy-free equilibrium	10, 5, 3.5, 2	slot 1: 300·(10 − 5) = 1500	**2400** = VCG

- In a lower slot A gets fewer clicks but pays less per click: 1000 > 600.
- **Locally envy-free equilibrium** (Edelman, Ostrovsky, Schwarz, 2007): no advertiser wants the slot just above at that slot's current price. B in slot 2 gets 200·(8 − 3.5) = 900; taking slot 1 at 5 would give 300·(8 − 5) = 900: no gain.

In these equilibria GSP revenue is at least 2400, and the lowest equals VCG. The 3600 from truthful bids is not an equilibrium revenue.

Budgets and Online Allocation

- Advertisers have daily **budgets**; searches arrive one by one, and the platform must assign each to an advertiser immediately, without knowing the future (an *online* decision).
- Example: A and B each have a 100 TL budget. Both bid 1 TL on “running shoes”; only A bids 1 TL on “gym bag”. First 100 shoe searches arrive, then 100 bag searches.

Rule	Shoe searches	Bag searches	Revenue
Greedy (highest bid, ties to A)	all 100 to A; A's budget runs out	A has no money left: 0	100
Balance: to the larger remaining budget	A 50, B 50	A's remaining 50	150
Optimum knowing the future	all 100 to B	all 100 to A	**200**

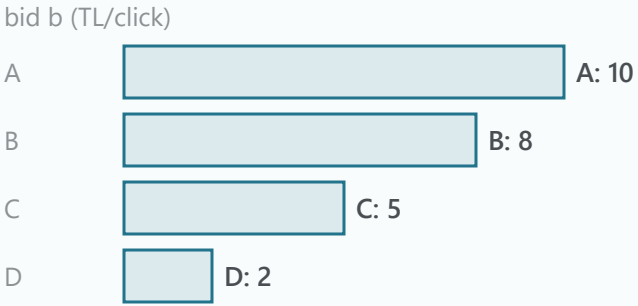
Common mistake

“Always picking the highest bid is best.” With budgets, the greedy rule exhausts the flexible advertiser early; in the worst case it guarantees only 1/2 of the optimum.

Guarantee

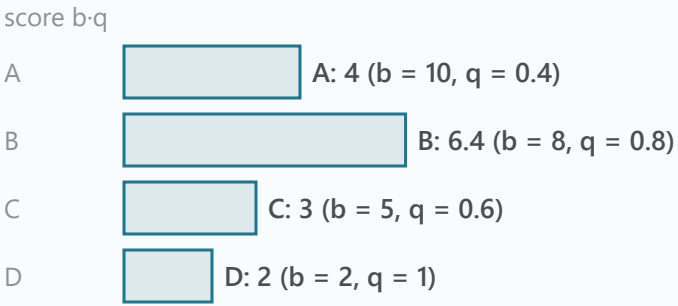
When bids are small relative to budgets, the MSVV rule — weighting bids by remaining budget — earns at least $1 - 1/e \approx 63\%$ of the optimum in any search order (Mehta, Saberi, Vazirani and Vazirani, 2007).

Step by Step: GSP with Quality Scores



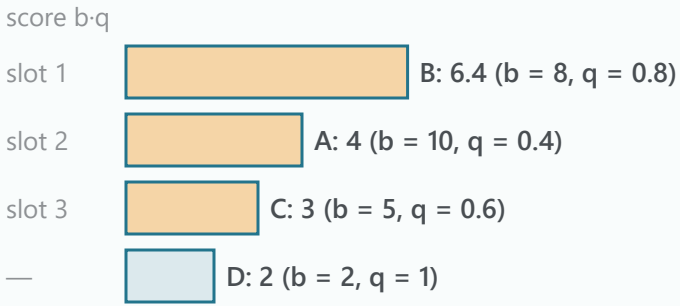
The same four advertisers. Clicks depend on slot and ad quality: i gets $\alpha_j \cdot q_i$ clicks in slot j (α = 300, 200, 100).

Step by Step: GSP with Quality Scores



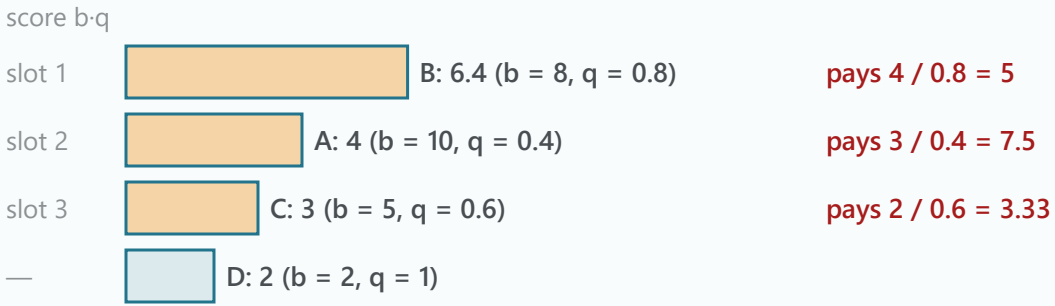
Quality q: how likely the ad is to be clicked (e.g. A 0.4, B 0.8). The platform ranks by the score **b·q**: expected revenue per impression.

Step by Step: GSP with Quality Scores



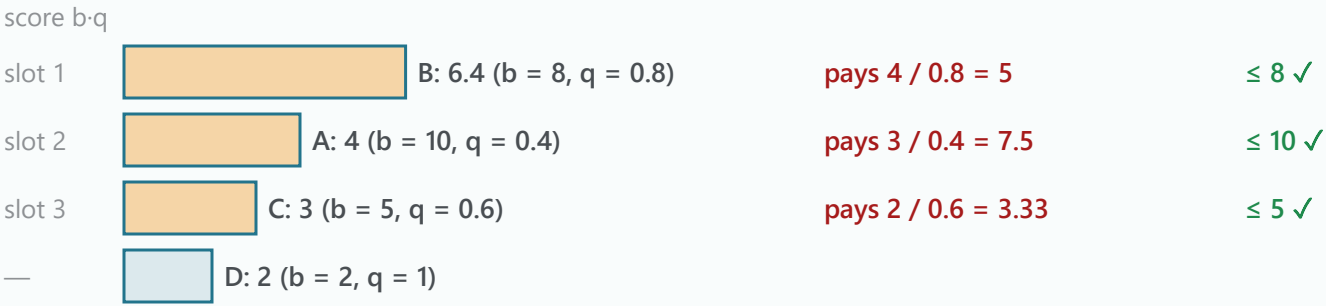
Ranked by score: B (6.4) > A (4) > C (3) > D (2). A, the highest bidder, is no longer first.

Step by Step: GSP with Quality Scores



Payment: the lowest bid that keeps the position. $b_i q_i \geq$ the score below $\Leftrightarrow b_i \geq \text{score}_{j+1}/q_i$.

Step by Step: GSP with Quality Scores



No payment exceeds the bid ($5 \le 8$, $7.5 \le 10$, $3.33 \le 5$). A high-quality ad gets the same slot for less: quality is rewarded.

Step by Step: GSP with Quality Scores

Ranking	slots 1–3	total clicks (per 1000 impressions)	revenue (TL)
By bid only	A, B, C	340	1,880
With quality scores (b·q)	B, A, C	380	2,000

Here quality scores raised both clicks (340 → 380) and revenue (1,880 → 2,000): users see relevant ads, and the platform sells more clicks.

Back to the Game: The Sponsored Search Auction

Theory's answer: no

With the "Search Ads" rates (12%, 7%, 3.5%), 1000 impressions give 120, 70, 35 clicks. Your click value is 8, rival bids 6, 4, 2, all qualities 1; GSP:

Bid	Slot	Payment/click	Utility
8 (truthful)	1	6	120 × 2 = 240
5	2	4	**70 × 4 = 280**
3	3	2	35 × 6 = 210

Dropping to a lower slot pays more; a best-response search finds exactly this deviation.

In practice

Under first-price ad auctions, bids repeatedly ratcheted up and crashed (Edelman and Ostrovsky, 2007). Under the GSP used by Google, truthful bidding is not dominant either (Edelman, Ostrovsky and Schwarz, 2007).

The name "second price" brings truthfulness with one slot, not with many. The rule that guarantees truthfulness is VCG.

Reputation Systems and Information Asymmetry

- Buyers don't know a seller's quality in advance: information asymmetry. Buyers then offer an average price, good sellers exit, and only bad products remain (Akerlof's “market for lemons”).
- A reputation system does two jobs: **selection** (find good sellers) and **discipline** (punish bad behavior in the future, a repeated game).

Fake reviews
Bought positive reviews, or negative ones for a rival.

Fear of retaliation
With mutual ratings, bad experiences go unreported.

Grade inflation
If everyone has 4.8+, ratings stop discriminating.

Selective participation
Moderately satisfied users stay silent; new sellers get no reviews at all (cold start).

Design measures: require verified purchases, down-weight old reviews, account for the uncertainty of few reviews (Bayesian averages), and reveal both sides' ratings simultaneously in mutual rating.

Step by Step: 5.0 from a Few Reviews or 4.7 from Many?

Product	reviews	average	rank	Bayesian score	rank
X	2	5	1		
Y	200	4.7	2		
Z	20	4.2	3		

Ranked by raw average: X (5.0) > Y (4.7) > Z (4.2). But X has only 2 reviews!

The average of a product with few reviews is very noisy; two fake reviews put it on top.

Step by Step: 5.0 from a Few Reviews or 4.7 from Many?

Product	reviews	average	rank	Bayesian score	rank
X	2	5	1	3.75	3
Y	200	4.7	2	4.64	1
Z	20	4.2	3	3.97	2

Bayesian average: $(C \cdot m + \Sigma \text{ ratings})/(C + n)$, with $m = 3.5$ (the platform mean), $C = 10$, n: the product's number of reviews. X: $(35 + 10)/12 = 3.75$; Y: $(35 + 940)/210 \approx 4.64$.

It is as if every product got 10 “average” virtual reviews: products with few reviews are pulled toward the mean. New ranking: Y > Z > X.

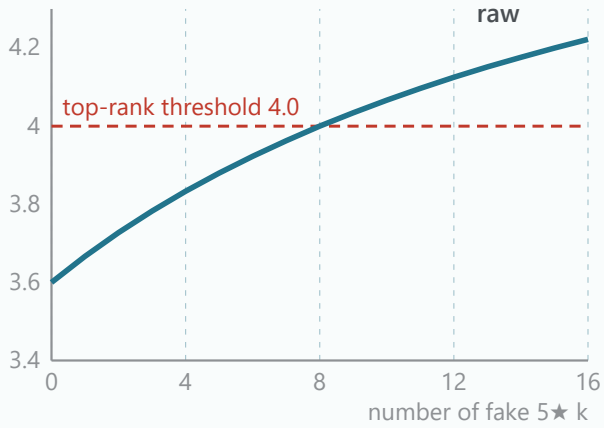
Step by Step: 5.0 from a Few Reviews or 4.7 from Many?

Product	reviews	average	rank	Bayesian score	rank
X + 3 fake	5	5	1	4	2
Y	200	4.7	2	4.64	1
Z	20	4.2	3	3.97	3

Adding 3 fake 5★ reviews to X keeps its raw average at 5.0 (still on top); its Bayesian score is (35 + 25)/15 = 4.0 → still below Y.

Bayesian averages blunt fake reviews: producing the same effect takes many more of them.

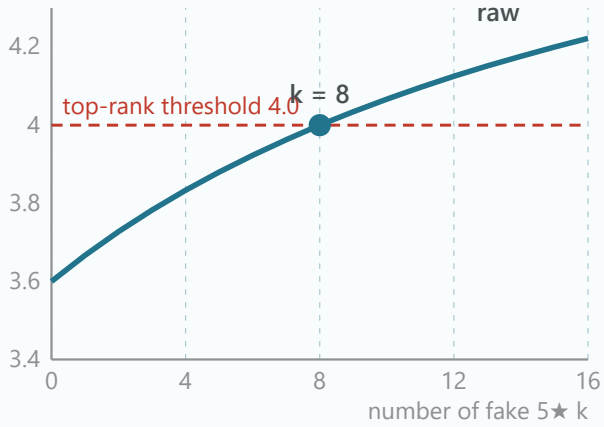
Step by Step: When Do Fake Reviews Pay?



A seller: 20 reviews, average 3.6. Ratings ≥ 4.0 rank at the top → extra monthly profit G = 600 TL. A fake 5★ review costs 15 TL.

Model the fake review as an investment decision: return, cost and expected penalty.

Step by Step: When Do Fake Reviews Pay?

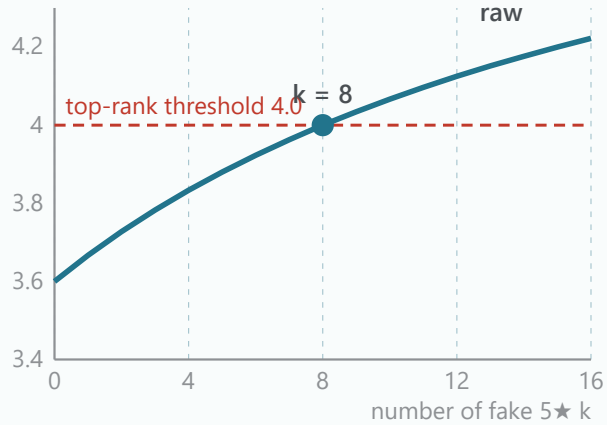


A seller: 20 reviews, average 3.6. Ratings ≥ 4.0 rank at the top → extra monthly profit G = 600 TL. A fake 5★ review costs 15 TL.

Raw average: $(72 + 5k)/(20 + k) \geq 4 \iff k \geq 8$. Cost $8 \times 15 = 120$ TL.

8 fake 5★ reviews lift the raw average to exactly 4.0.

Step by Step: When Do Fake Reviews Pay?



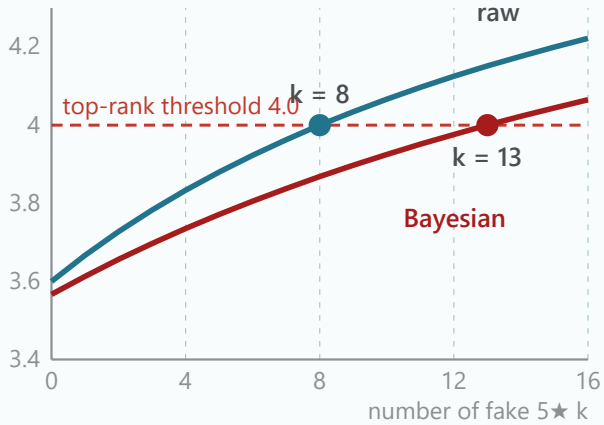
A seller: 20 reviews, average 3.6. Ratings ≥ 4.0 rank at the top → extra monthly profit G = 600 TL. A fake 5★ review costs 15 TL.

Raw average: (72 + 5k)/(20 + k) ≥ 4 ⟺ k ≥ 8. Cost 8 × 15 = 120 TL.

Detection probability ρ; if caught, loss L = 3000 TL. Profitable ⟺ 600 − 120 > ρ·3000 ⟺ **ρ < 0.16**.

Expected gain 600 − 120 − ρ·3000. If the platform's detection probability is below 16%, cheating pays.

Step by Step: When Do Fake Reviews Pay?



A seller: 20 reviews, average 3.6. Ratings ≥ 4.0 rank at the top → extra monthly profit G = 600 TL. A fake 5★ review costs 15 TL.

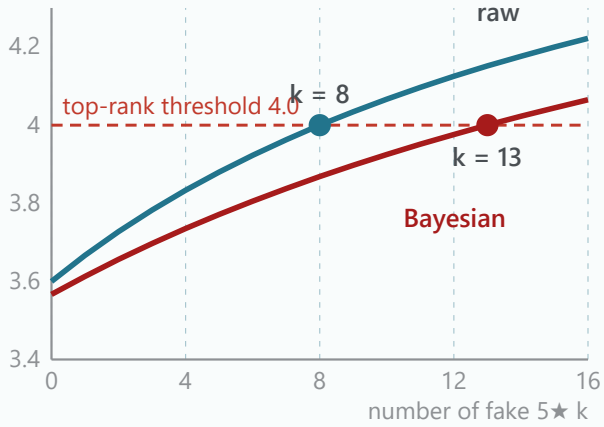
Raw average: $(72 + 5k)/(20 + k) \geq 4 \Leftrightarrow k \geq 8$. Cost 8 × 15 = 120 TL.

Detection probability ρ; if caught, loss L = 3000 TL. Profitable $\Leftrightarrow 600 - 120 > \rho \cdot 3000 \Leftrightarrow$ **$\rho < 0.16$**.

With the Bayesian score (m = 3.5, C = 10): $(107 + 5k)/(30 + k) \geq 4 \Leftrightarrow k \geq 13$; cost 195 TL → **$\rho < 0.135$**.

Reaching the same threshold under the Bayesian score takes 13 fakes: cheating costs more, and the break-even drops to $\rho < 0.135$.

Step by Step: When Do Fake Reviews Pay?



A seller: 20 reviews, average 3.6. Ratings ≥ 4.0 rank at the top → extra monthly profit G = 600 TL. A fake 5★ review costs 15 TL.

Raw average: $(72 + 5k)/(20 + k) \geq 4 \iff k \geq 8$. Cost 8 × 15 = 120 TL.

Detection probability ρ; if caught, loss L = 3000 TL. Profitable $\iff 600 - 120 > \rho \cdot 3000 \iff$ **$\rho < 0.16$**.

With the Bayesian score (m = 3.5, C = 10): $(107 + 5k)/(30 + k) \geq 4 \iff k \geq 13$; cost 195 TL → **$\rho < 0.135$**.

Design levers: raise ρ (verified purchases, anomaly detection), raise L (account bans), lower G (robust scoring).

The mechanism-design view: the platform doesn't just ban cheating, it pushes **the expected return of cheating below zero**.

Platform Competition and Regulation

Competition

- **Winner takes all**: through tipping, a single platform can capture most of the market.
- **Switching costs**: friends, past ratings, data → keep users locked in.
- **Multi-homing**: using several platforms at once (a driver on both Uber and a rival) weakens monopolization.
- Examples: Hepsiburada – Trendyol, Uber – Lyft, Instagram – TikTok.

Ethics and regulation

- Data privacy and security.
- Monopolization risk and self-preferencing.
- Transparency of ranking and recommendation algorithms.
- Example: the EU Digital Markets Act (DMA) imposes obligations on large “gatekeeper” platforms.

Recommender Systems, Lock-in and Privacy

- Recommendation and ranking algorithms **allocate visibility** (who gets featured?): they are an allocation rule, and participants choose strategies accordingly.

The platform's objective
Maximizing short-term engagement or revenue can conflict with users' long-term interest (clickbait, infinite scroll).

Creators are players too
Creators and sellers compete for visibility and converge on whatever form the algorithm rewards (titles, tags): content can become uniform.

Loyalty programs
Loyalty points, paid memberships (e.g. fast shipping), gamification (badges, streaks): deliberately raise switching costs and reduce multi-homing.

Data and privacy
The price of a “free” service is data: more data means better recommendations but also risks of leaks and misuse. Users choose strategically how much to share.

The mechanism-design question is the same: are the platform's goals aligned with the interests of users and creators?

Discussion: Growth, Behavior and Fairness

Why do some platforms grow fast while others collapse?

Critical mass and tipping: if expectations are above the threshold, feedback drives growth; below it, collapse. Ease of use, trust and early incentives help clear the threshold.

How do platforms shape behavior?

Bonuses and discounts boost specific behaviors (driver bonuses in busy areas); recommendation algorithms steer preferences; spam and fake-review policing limits manipulation.

Can a design be fair to everyone?

Perfect fairness is hard: preferences differ, network effects favor large groups, and players manipulate strategically. Incentives, rules and regulation can improve things.

ASSESSMENT

Summary

Concept	Explanation	Examples
Network effects, tipping	$F(x) = 1 - p/x$; two stable equilibria with a threshold between them	$p = 0.16$: 0 and 0.8 stable, threshold 0.2
Two-sided pricing	$p_A = n_A - e_B n_B$: the side that adds more value to the other pays less	$p_A = -5$, $p_B = 50$, $\Pi = 2800$
GSP	Slot j pays the next bid down; truthful bidding is not dominant	revenue 3600 with the same bids; A bids 6 instead of 10
VCG	Pays the loss imposed on others; truthful bidding is dominant	1500 / 700 / 200, revenue 2400
GSP equilibrium	The lowest locally envy-free equilibrium gives the VCG outcome	bids 10, 5, 3.5, 2 → 5 / 3.5 / 2 TL/click
Reputation	Bayesian averages make fake reviews more expensive	fake reviews needed: 8 → 13

ASSESSMENT

Sample Problems

1. In the adoption model let $p = 0.21$. What are the equilibria and the critical mass?
2. In the marketplace model let $e_A = e_B = 0.5$ (all else equal). What are the optimal prices? Is there a subsidy?
3. Two slots (100 and 60 clicks), bids 6, 4, 1. Find the total GSP and VCG payments.
4. In the GSP of problem 3, with the others truthful, should the advertiser valued at 6 bid truthfully?
5. 10 reviews, average 4.0. How many fake 5★ reviews raise the raw average to 4.5?
6. Two slots; A bids 6 ($q = 0.5$), B 4 ($q = 1$), C 1 ($q = 1$). In GSP with quality scores, what are the ranking and per-click payments?

▶ **Show answers**

References

- J.-C. Rochet, J. Tirole, “Platform Competition in Two-Sided Markets”, *Journal of the European Economic Association*, 2003.
- M. Armstrong, “Competition in Two-Sided Markets”, *RAND Journal of Economics*, 2006.
- B. Edelman, M. Ostrovsky, M. Schwarz, “Internet Advertising and the Generalized Second-Price Auction”, *American Economic Review*, 2007.
- H. Varian, “Position Auctions”, *International Journal of Industrial Organization*, 2007.
- D. Easley, J. Kleinberg, ***Networks, Crowds, and Markets***, Cambridge University Press, 2010 (Chapters 15 and 17).
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (eds.), *Algorithmic Game Theory*, Cambridge University Press, 2007 (Chapters 27–28).
- T. Roughgarden, **Stanford CS364A: Algorithmic Game Theory** lecture notes, 2013 (Lectures 2–3: sponsored search auctions).
- G. Akerlof, “The Market for ‘Lemons’”, *Quarterly Journal of Economics*, 1970.
- M. Luca, G. Zervas, “Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud”, *Management Science*, 2016.

References: Ad Auctions

- H. Varian, C. Harris, “The VCG Auction in Theory and Practice”, *American Economic Review: Papers & Proceedings* 104(5), 2014.
- B. Edelman, M. Ostrovsky, “Strategic Bidder Behavior in Sponsored Search Auctions”, *Decision Support Systems* 43(1), 2007. · M. Ostrovsky, M. Schwarz, “Reserve Prices in Internet Advertising Auctions: A Field Experiment”, *ACM EC*, 2011.
- A. Mehta, A. Saberi, U. Vazirani, V. Vazirani, “AdWords and Generalized Online Matching”, *Journal of the ACM* 54(5), 2007.

The End

Chapter 6: Online Platforms

Previous: Chapter 5: Auctions · Next: Chapter 7: Social Choice and Voting