Envy-free economic allocations

Visualizing an algorithm from an economics paper

Justin Pearson
2018-10-20

Summary

We illustrate an algorithm from an economics paper by a Dutch economics professor named Flip Klijn. The paper addresses the problem of allocating n objects to n people (each person gets 1 object). Each person derives some amount of utility ("happiness") from each object. After allocating the objects, some people may envy others, e.g., Alice's utility would be higher if she had been allocated Bob's object instead of her own. Indeed, depending on people's preferences for the objects, it may not be possible to allocate the objects in an envy-free way: For example, if we have 1 ferrari and 9 goats to allocate, everyone will envy the ferrari owner. The paper considers the following idea: Suppose whoever got the ferrari had to give $10,000 to each person who got a goat. This sweetens the goat prize and makes the ferrari prize less attractive: Who wants a ferrari if you also have to pay $90,000? I would rather own a goat and $10,000 than a ferrari and -$90,000. If enough money is transferred from the ferrari-owner to the goat-owners, the goat-owners wouldn't envy the ferrari owner. The paper presents an algorithm to allocate objects and transfer money between people so that everyone is happy with his or her allocation of object and money.

Source

"An Algorithm for Envy-free Allocations in an Economy with Indivisible Objects and Money"
Flip Klijn, Department of Econometrics and CentER, Tilburg University, The Netherlands
https://pdfs.semanticscholar.org/1722/1cbf08f9a28e527fb9037e96260f4da73345.pdf

Outline

A set $Q := \{1, \ldots, n\}$ of n indivisible objects and an amount $M \geq 0$ of money are to be distributed among a set $N := \{1, \ldots, n\}$ of agents.
Utility: Agents derive utility (happiness) from objects and from money: U_{ij} is how much agent i likes object j and α_i is agent i's utility per unit money. (We assume each agent's money-utility is linear.)
Allocation: Each agent is assigned an object and an amount of money: Agent i gets object σ_i and money x_i, and therefore derives an amount of utility $U_{i\,\sigma_i} + \alpha_i x_i$. The object-allocation σ is a bijection

from agents to objects, and the money-allocation $x \in \mathbb{R}^n$ has $\Sigma x_i = M$.

Envy: Given an allocation (σ, x), the quantity $e(i, j) := \left(U_{i\,\sigma_j} + \alpha_i x_j\right) - (U_{i\,\sigma_i} + \alpha_i x_i)$ is the amount of "envy" that agent i feels for agent j.

- The first term is how much agent i likes agent j's loot. The second term is how much agent i likes his own loot.

- If $e(i, j) > 0$, then agent i would have higher utility if he had agent j's loot instead.

- If $e(i, j) == 0$, then agent i would keep the same utility if he had j's loot instead.

- If $e(i, j) < 0$, then agent i would have lower utility if he had agent j's loot instead.

Goal: find an allocation (σ, x) where no agent envies any other agent: $e(i, j) \leq 0$ for all $i, j \in N$.

Envy graph: A graph where agents are vertices and we have a directed edge $i \to j$ if $e(i, j) \geq 0$. If $e(i, j) > 0$, $i \to j$ is called a *strong edge*. If $e(i, j) == 0$ it's called a *weak edge*.

Algorithm: Let a feasible allocation (σ, x) be given. Consider its envy-graph.

Step 1. If there are no strong arcs, then we have an envy-free allocation. Stop. Otherwise go to step 2.

Step 2. If there is a cycle containing a strong arc, then apply the **permutation procedure** and go to step 1. Otherwise pick a strong arc and go to step 3.

Step 3. Apply the **sidepayment procedure** to the picked arc and go to step 4.

Step 4. If a strong arc has been eliminated, go to step 1. If a cycle with a strong arc has emerged, go to step 2. Otherwise go to step 3.

Permutation procedure: You've got a cycle like $i \to j \to k \to i$. Allocate objects and money like $i \leftarrow (\sigma_j, x_j)$, $j \leftarrow (\sigma_k, x_k)$, $k \leftarrow (\sigma_i, x_i)$. Now everyone has an allocation they like better.

Sidepayment procedure: You've got a strong edge $i \to j$. Let P denote the agents for which a weak-edged path to i exists, and let R denote agents who are reachable along a weak-edged path starting from j. Take money from R and give it to P until a weak edge forms.

Discussion

In[1]:=
```
Clear["Global`*"]
SetDirectory[NotebookDirectory[]];
str[args__] := ({args} // Map[ToString] // StringRiffle // StringJoin)
```

A set $Q := \{1, \ldots, n\}$ of n indivisible objects and an amount $M \geq 0$ of money are to be distributed among a set $N := \{1, \ldots, n\}$ of agents.

In[4]:=
```
imPreprocess[im_Image] := Module[{im2, bg, alpha},
  im2 = ImageResize[im, 70];
  bg = RemoveBackground[im2, {White, .001}];
  alpha = bg // AlphaChannel // Dilation[#, 1] & // FillingTransform // Binarize;
  SetAlphaChannel[im2, alpha]
 ]
SetAttributes[imPreprocess, Listable];
```

```
agentPix = {
```





```
} // imPreprocess;
```

```
objectPix = {
```





```
} // imPreprocess;
```

```
showAgent[agent_] := Show[agentPix[[agent]], ImageSize → 40] // Overlay[{#, agent}] &;
showObject[object_] := Show[objectPix[[object]], ImageSize → 40] // Row[{object, #}] &;
SetAttributes[showAgent, Listable];
SetAttributes[showObject, Listable];
n = Length@agentPix;
M = 0;(* Paper says you can always rescale utilities to make M=0 WLOG *)
objects = Range[n];
agents = Range[n];
```

Utility

Utility: Agents derive utility (happiness) from objects and from money: U_{ij} is how much agent i likes object j, and α_i is agent i's utility per unit money. (We assume each agent's money-utility is linear.)

In[16]:=
```
U = {{60 000, 100, 0, 0, 0},
     {80 000, 70, 200, 200, 200},
     {20 000, 1000, 0, 0, 0},
     {25 000, 900, 0, 0, 0},
     {30 000, 1100, 1000, 1000, 1000}};

α = {2/10, 5/10, 1, 2, 6} // Flatten;(* Note: Must be rational to keep inf-precision. *)

Grid[{{
   TableForm[U, TableHeadings → {showAgent@agents, showObject@objects}] // Framed //
    Labeled[#, "Object utilities.", Top] &,
   TableForm[{#} & /@ α, TableHeadings → {showAgent@agents, None}] // Framed //
    Labeled[#, "Utility per money.", Top] &
  }}, Frame → True, Alignment → Top, Spacings → 2 {1, 1}]
```

Out[17]=

Object utilities.

	1	2	3	4	5
1	60 000	100	0	0	0
2	80 000	70	200	200	200
3	20 000	1000	0	0	0
4	25 000	900	0	0	0
5	30 000	1100	1000	1000	1000

Utility per money.

1	1/5
2	1/2
3	1
4	2
5	6

In[18]:=
```
util[agent_, object_, money_] := U[[agent, object]] + α[[agent]] * money
```

Here is how much agent 1 likes object 2 and $100:

In[19]:=
```
util[1, 2, 100]
```

Out[19]=
```
120
```

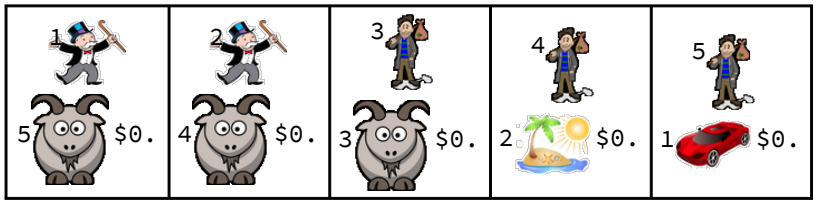
Allocation of objects & money

Allocation: Each agent is assigned an object and an amount of money: Agent i gets object σ_i and money x_i, and therefore derives an amount of utility $U_{i\,\sigma_i} + \alpha_i x_i$. The object-allocation σ is a bijection from agents to objects, and the money-allocation $x \in \mathbb{R}^n$ has $\Sigma x_i = M$.

In[20]:=
```
initializeAllocation := (
  σ = Reverse@objects;
  x = ConstantArray[M / n, n];
 )
initializeAllocation
show[agent_] := Column[{showAgent[agent],
   Row[{showObject[σ[[agent]]], "$" <> ToString@Round[x[[agent]], .01]}, " "]}, Center]
Print["Initial allocation:"];
show /@ agents // Grid[{#}, Frame → All] &
showAllocationTable[title_String] :=
 TableForm[{agents, σ, x, MapThread[util, {agents, σ, x}]}^T,
    TableHeadings → {None, {"agent", "object", "money", "utility"}}] //
   Labeled[#, title, Top] & // Framed
showAllocationTable["Initial allocation of objects and money."]
```

```
Initial allocation:
```

Out[24]=



Out[26]=

Initial allocation of objects and money.

agent	object	money	utility
1	5	0	0
2	4	0	200
3	3	0	0
4	2	0	900
5	1	0	30 000

In[27]:=
```
totalUtility := Total@MapThread[util, {agents, σ, x}]
```

In[28]:=
```
totalUtility
```

Out[28]= 31 100

Envy

Envy: Given an allocation (σ, x), the quantity $e(i, j) := \left(U_{i\,\sigma_j} + \alpha_i\, x_j\right) - \left(U_{i\,\sigma_i} + \alpha_i\, x_i\right)$ is the amount of "envy" that agent *i* feels for agent *j*.

In[29]:=
```
envy[i_, j_] := util[i, σ[[j]], x[[j]]] - util[i, σ[[i]], x[[i]]]
```

- The first term is how much agent *i* likes agent *j*'s loot. The second term is how much agent *i* likes his own loot.

- If $e(i, j) > 0$, then agent *i* would have higher utility if he had agent *j*'s loot instead.

- If $e(i, j) == 0$, then agent *i* would keep the same utility if he had *j*'s loot instead.

- If $e(i, j) < 0$, then agent *i* would have lower utility if he had agent *j*'s loot instead.

Table of envies:

In[30]:=
```
showEnvyTable[title_String] := Table[
        With[{e = envy[i, j]}, Style[e, If[e > 0, Red, Black]]],
        {i, agents}, {j, agents}
       ] //
       Grid[#, Frame → All] & //
      Legended[#, SwatchLegend[{Red}, {"envy > 0"}]] & //
     Labeled[#, title, Top] & //
    Framed //
  Print
```

In[31]:=
```
showEnvyTable["The initial allocation is not envy-free."]
```

The initial allocation is not envy-free.

0	0	0	100	60 000
0	0	0	−130	79 800
0	0	0	1000	20 000
−900	−900	−900	0	24 100
−29 000	−29 000	−29 000	−28 900	0

envy > 0

Problem Statement

Goal: find an allocation (σ, x) where no agent envies any other: $e(i, j) \leq 0$ for all $i, j \in N$.

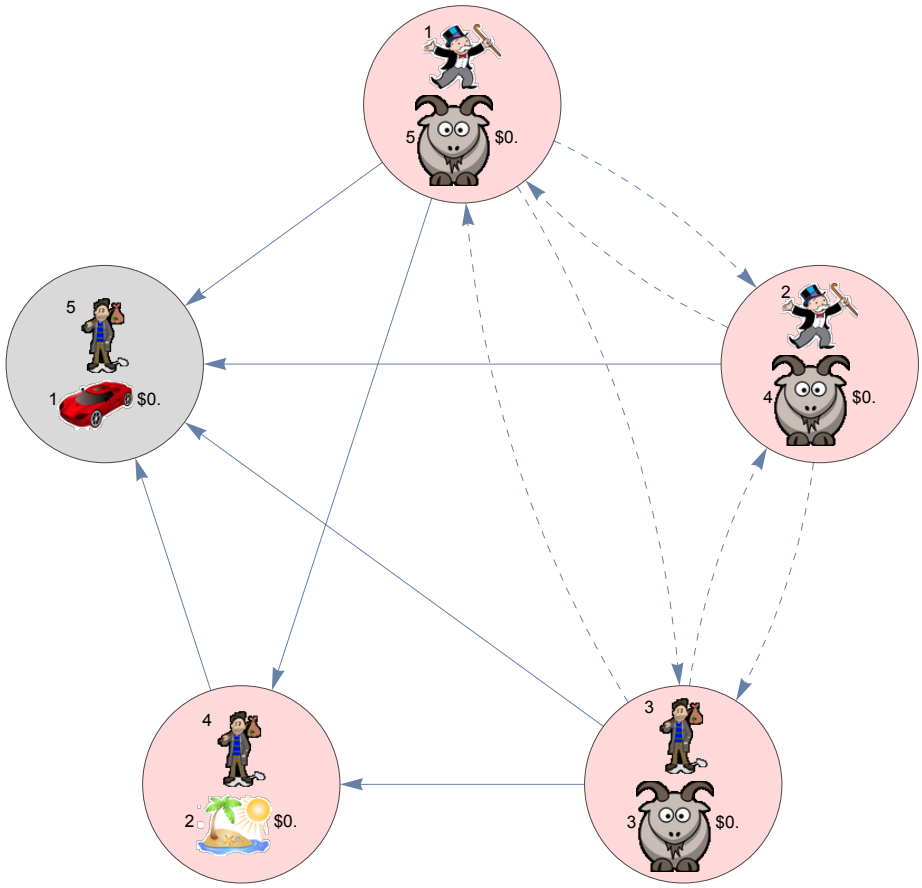
Envy Graph

Envy graph: A graph where agents are vertices and $i \to j$ if $e(i, j) \geq 0$. If $e(i, j) > 0$, $i \to j$ is called a *strong edge*. If $e(i, j) == 0$ it's called a *weak edge*.

In[32]:=
```
envyGraph[type_String: "strong and weak edges"] :=
 With[{f = Switch[type, "strong edges", Positive, "weak edges", # == 0 &, "strong and weak edges", NonNegative]},
  RelationGraph[
   #1 ≠ #2 && f[envy[#1, #2]] &,
   agents,
   DirectedEdges → True,
   VertexLabels → Table[i → Placed[i, Center, show], {i, agents}],
   VertexSize → {"Scaled", .2},
   VertexCoordinates → ({Re@#, Im@#} & /@ ℯ^(i (-2*π* (agents-1)/n + 90 Degree))),
   EdgeStyle → {(i_ ↔ j_ /; (envy[i, j] == 0)) → Dashed},
   VertexStyle →
    Table[i → Switch[AnyTrue[agents, envy[i, #] > 0 &], True, LightRed, _, LightGray], {i, agents}],
   ImageSize → 495 (*500: frame no appear in PDF*)
  ]]
```

In[33]:=
```
envyGraph[]
```

Out[33]=



Useful functions for working the the envy graph:

In[34]:=
```
strongEdges := envyGraph[] // EdgeList // Select[Positive@*Apply[envy]]
```

In[35]:=
```
envyGraphCycles := envyGraph[] // FindCycle[#, ∞, All] &
```

In[36]:=
```
getCycleWithStrongEdge := SelectFirst[envyGraphCycles, ContainsAny[strongEdges], {}]
```

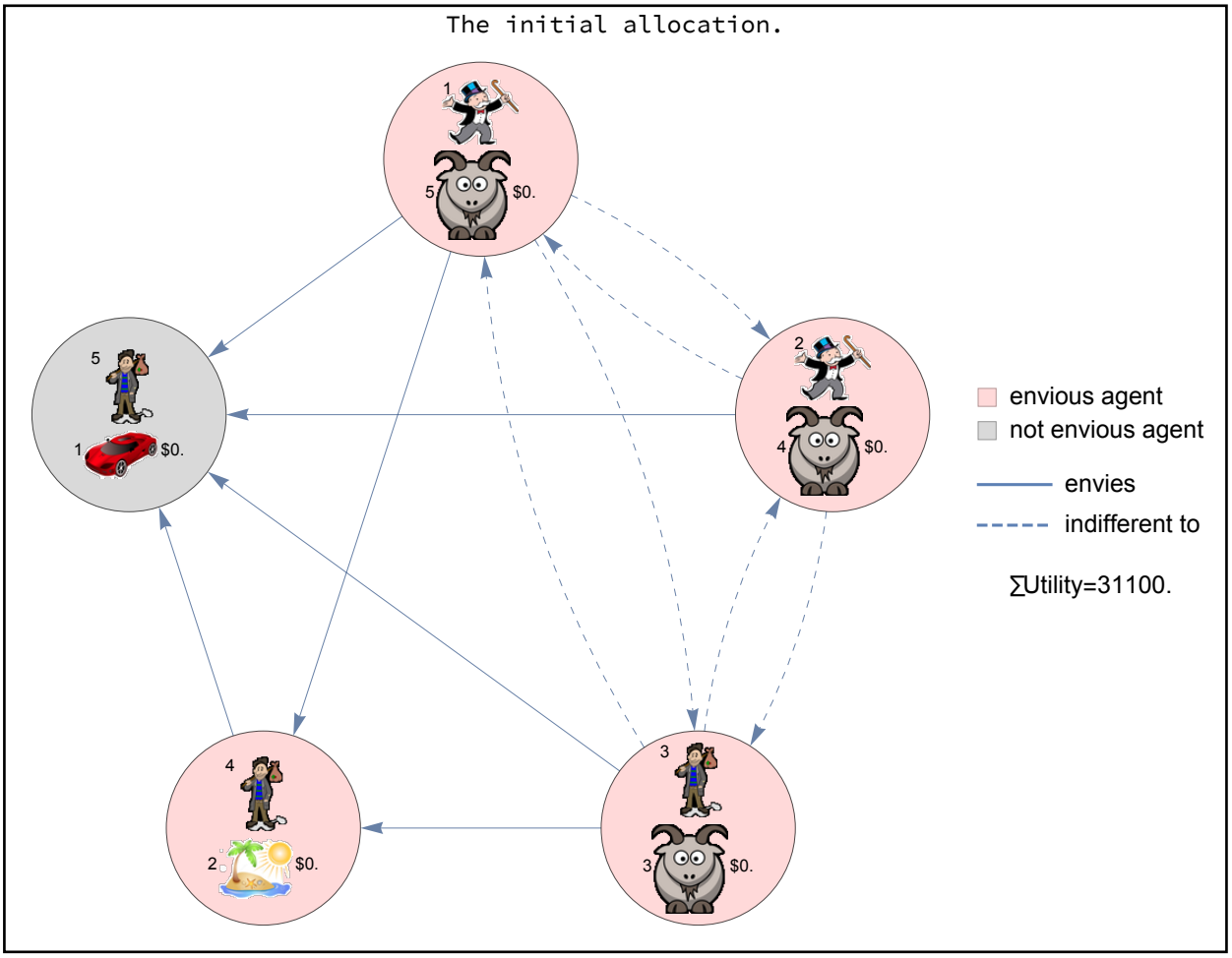
In[37]:=
```
hasCycleWithStrongEdge := getCycleWithStrongEdge ≠ {}
```

In[38]:=
```
styleEdge[e_, color_: Darker[Red]] :=
 Style[e, Thick, color, If[envy @@ e == 0, Dashed, Nothing]]
SetAttributes[styleEdge, Listable]
```

In[40]:=
```
addLegends[g_, otherLegs_: {}] := Block[{leg1, leg2, leg3},
   leg1 =
    SwatchLegend[{LightRed, LightGray}, {"envious agent", "not envious agent"}];
   leg2 = With[{c = ColorData[97, "ColorList"][[1]]},
     LineLegend[{c, {c, Dashed}}, {"envies", "indifferent to"}]];
   leg3 = PointLegend[{White}, {"∑Utility=" <> ToString[N@totalUtility]}];
   Legended[g, Flatten@{{leg1, leg2, leg3}, otherLegs}]];
```

In[41]:=
```
envyGraph[] // addLegends // Labeled[#, "The initial allocation.", Top] & // Framed
```

Out[41]=



Next we discuss two procedures that will be used in the overall algorithm.

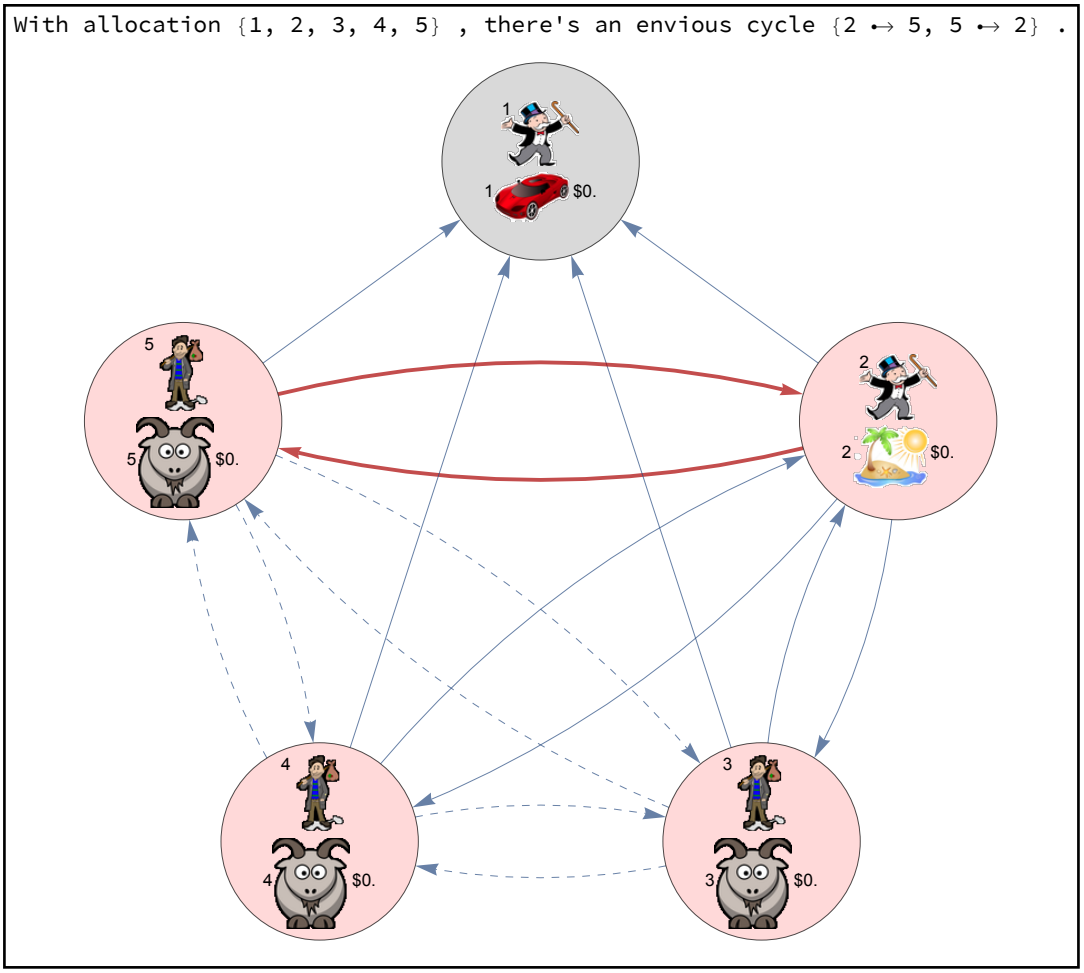
Permutation Procedure

Permutation procedure: You've got a cycle like $i \to j \to k \to i$. Allocate objects and money like $i \leftarrow (\sigma_j, x_j)$, $j \leftarrow (\sigma_k, x_k)$, $k \leftarrow (\sigma_i, x_i)$. Now everyone has an allocation they like better.

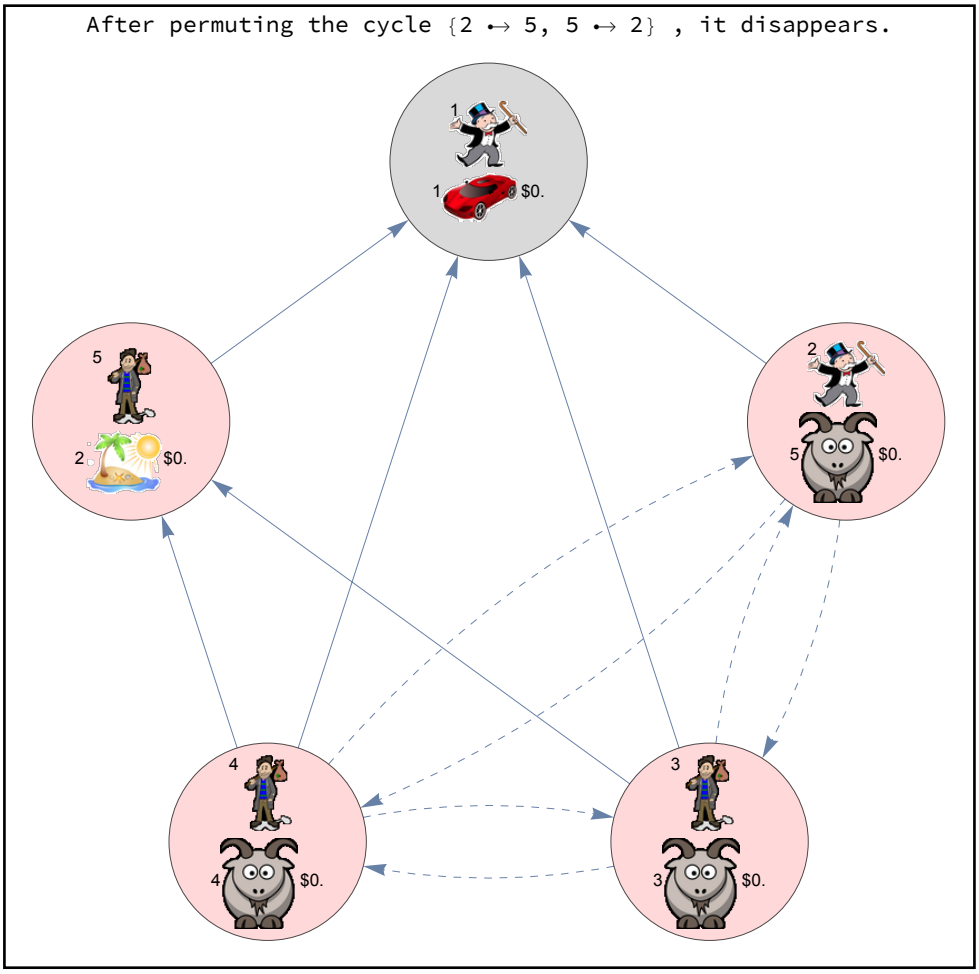
In[42]:=
```
permutationProcedure[cycle_] := (
    Print["Permuting..."];
    σ[[cycle[[All, 1]]]] = σ[[cycle[[All, 2]]]];
    x[[cycle[[All, 1]]]] = x[[cycle[[All, 2]]]]
  );
```

Example

In[43]:=
```
Block[{σ = Range[n], x = 0 * Range[n], c},
 c = getCycleWithStrongEdge;
 Labeled[HighlightGraph[envyGraph[], styleEdge@c], str["With allocation",
     σ, ", there's an envious cycle", c, "."], Top] // Framed // Print;
 permutationProcedure[c];
 Labeled[envyGraph[], str["After permuting the cycle", c, ", it disappears."],
    Top] // Framed // Print;
]
```



Permuting...



Sidepayment Procedure

Sidepayment procedure: You've got a strong edge $i \to j$. Let P denote the agents for which a weak-edged path to i exists, and let R denote agents who are reachable along a weak-edged path starting from j. Take money from R and give it to P until a weak edge forms.

Analysis

We now determine the least amount of money to transfer for a weak edge to form.

Let $t > 0$ denote the total amount of money that will be transferred from R to P. Let $r \in \mathbb{R}^n$ be defined as

$$r_i = \begin{cases} \frac{1}{|P|} & i \in P \\ -\frac{1}{|R|} & i \in R \\ 0 & \text{else} \end{cases}$$

so that $r_i\, t$ is the amount of money given to agent i for $i = 1, \ldots, n$. For any two agents i, j, the value of t that results in $e(i, j) == 0$ — a weak edge from i to j — satisfies

$$\begin{aligned} 0 &= e(i, j) \\ &= \left(U_{i\,\sigma_j} + \alpha_i (x_j + r_j\, t)\right) - (U_{i\,\sigma_i} + \alpha_i (x_i + r_i\, t)) \\ &= \left(U_{i\,\sigma_j} - U_{i\,\sigma_i}\right) + \alpha_i ((x_j - x_i) + t\,(r_j - r_i)) \\ &= \Delta U_{ij} + \alpha_i (\Delta x_{ij} + t\, \Delta r_{ij}), \end{aligned}$$

and solving this for t yields

$t = -\frac{\Delta U_{ij}/\alpha_i + \Delta x_{ij}}{\Delta r_{ij}}$.

Note that we only have $t > 0$ in certain cases, for example when $i \in R$ and $j \in P$. There are some cases, e.g., when i, j belong to the same class, for which $\nexists\, t$ for which $e(i, j) == 0$.

Therefore, the smallest t for which a weak edge appears is

$\min_{i,j \in N} \left\{ -\frac{\Delta U_{ij}/\alpha_i + \Delta x_{ij}}{\Delta r_{ij}} \right\}$,

where we only Min over the finite, positive values.

Code

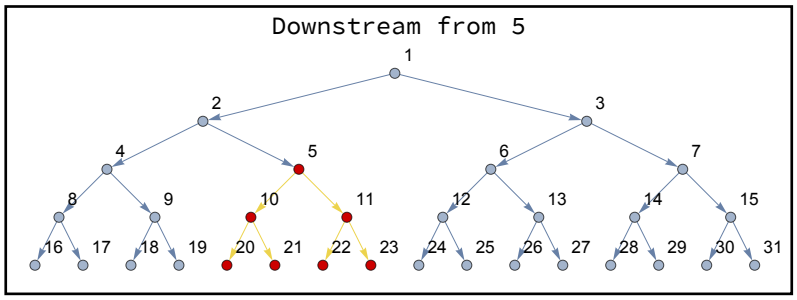
In[44]:=
```
downstreamFrom[g_, v_] :=
 Reap[BreadthFirstScan[g, v, {"DiscoverVertex" → (Sow[{#1, #2 ↦ #1}] &)}]][[2, 1]] //
  Transpose
```

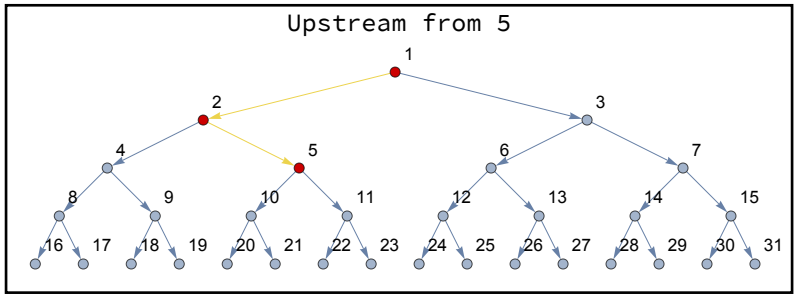
In[45]:=
```
upstreamFrom[g_, v_] := Reap[BreadthFirstScan[ReverseGraph@g,
      v, {"DiscoverVertex" → (Sow[{#1, #1 ↦ #2}] &)}]][[2, 1]] // Transpose
```

In[46]:=
```
Block[{g = CompleteKaryTree[5,
    DirectedEdges → True, VertexLabels → Automatic, ImageSize → 400]},
 Row[Framed /@ {
    HighlightGraph[g, downstreamFrom[g, 5]] //
     Labeled[#, "Downstream from 5", Top] &,
    HighlightGraph[g, upstreamFrom[g, 5]] // Labeled[#, "Upstream from 5", Top] &
   }] //
  Labeled[#, "Example of downstreamFrom[] and upstreamFrom[] functions.", Top] &
]
```

Out[46]=

Example of downstreamFrom[] and upstreamFrom[] functions.





In[47]:=
```
leastMoneyForNewWeakEdge[poorAgents_, richAgents_] :=
 Module[{r, moneyTable, leastMoney, newWeakEdge},
  r = Table[Piecewise[{{1/Length@poorAgents, MemberQ[poorAgents, i]},
                       {-1/Length@richAgents, MemberQ[richAgents, i]}},
                      0, True], {i, agents}];
  moneyTable = Quiet@Table[With[{
       ΔU = U[[i, σ[[j]]]] - U[[i, σ[[i]]]],
       Δx = x[[j]] - x[[i]],
       Δr = r[[j]] - r[[i]]
      },
      -(ΔU / α[[i]] + Δx)/Δr], {i, agents}, {j, agents}];
  leastMoney = moneyTable // Flatten // Select[Positive] // Min;
  newWeakEdge = DirectedEdge @@ FirstPosition[moneyTable, leastMoney];
  Return[{leastMoney, newWeakEdge}]
 ]
```

In[48]:=
```
transferMoney[m_, from_, to_] := (
  x[[to]] += 1/Length@to * m;
  x[[from]] -= 1/Length@from * m;
 )
```

Computing the data relevant to the sidepayment before actually performing it allows us to display it without actually doing it.

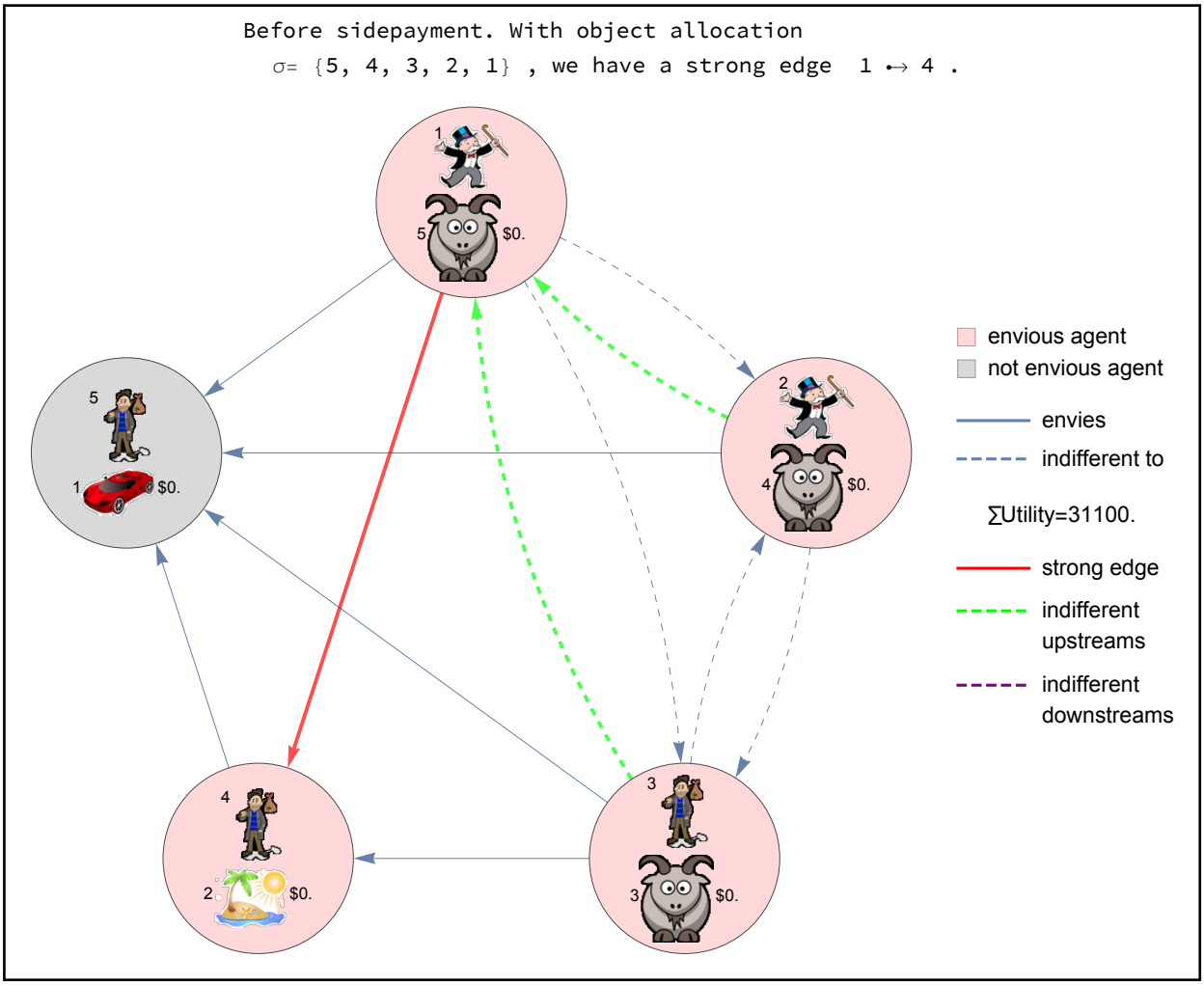
In[49]:=
```
sidepaymentData[e_DirectedEdge] :=
 Module[{poorAgent = e[[1]], richAgent = e[[2]], poorAgentEdges,
   poorAgents, richAgentEdges, richAgents, leastMoney, newWeakEdge},
  {poorAgents, poorAgentEdges} = upstreamFrom[envyGraph["weak edges"], poorAgent];
  {richAgents, richAgentEdges} =
   downstreamFrom[envyGraph["weak edges"], richAgent];
  {leastMoney, newWeakEdge} = leastMoneyForNewWeakEdge[poorAgents, richAgents];

  ⎛          "strong edge of interest"                 e          ⎞
  ⎜                "poor agent"                     poorAgent     ⎟
  ⎜                "rich agent"                     richAgent     ⎟
  ⎜     "indifferent upstream poor agents"         poorAgents     ⎟
  ⎜  "indifferent upstream poor agent edges"   poorAgentEdges     ⎟ //
  ⎜    "indifferent downstream rich agents"       richAgents     ⎟
  ⎜ "indifferent downstream rich agent edges"  richAgentEdges     ⎟
  ⎜      "least money for new weak edge"          leastMoney     ⎟
  ⎝               "new weak edge"                 newWeakEdge    ⎠

   Apply[Rule, #, {1}] & // Association
 ]
```

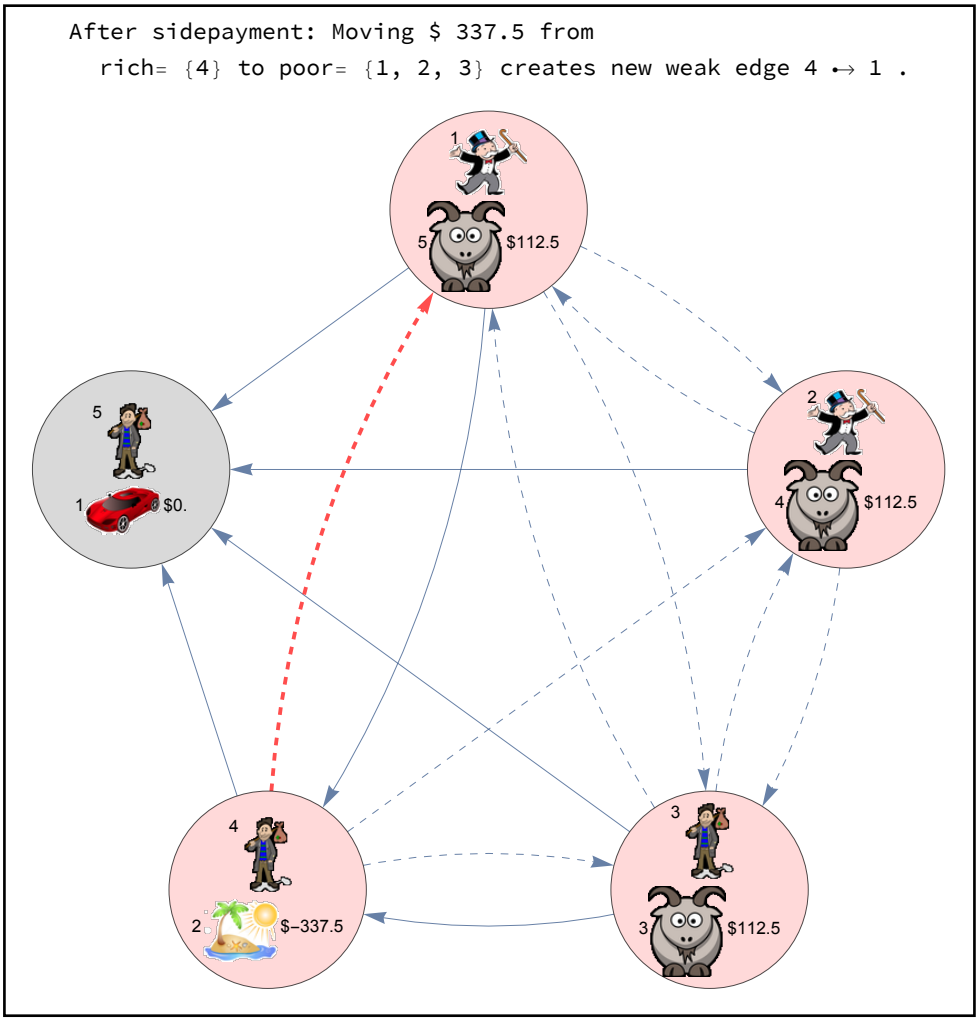
Here is the function that performs the **sidepayment procedure**.

In[50]:=
```
sidepaymentProcedure[e_DirectedEdge] := With[{d = sidepaymentData[e]},
  transferMoney[
    d["least money for new weak edge"],
    d["indifferent downstream rich agents"],
    d["indifferent upstream poor agents"]
   ];
 ]
```

Example of sidepayment procedure

In[51]:=
```
step3Styles[e_DirectedEdge,
  poorEdges : {__DirectedEdge}, richEdges : {__DirectedEdge}] :=
 Flatten@{Style[e, Thick, Red],
   Style[poorEdges, Thick, Green, Dashed],
   Style[richEdges, Thick, Purple, Dashed]}

step3Legend = LineLegend @@
   Transpose@{{Red, "strong edge"},
     {Directive[Green, Dashed], "indifferent\nupstreams"},
     {Directive[Purple, Dashed], "indifferent\ndownstreams"}};


Block[{σ, x, d, e},
 initializeAllocation;
 SeedRandom[124]; e = RandomChoice@strongEdges;
 d = sidepaymentData[e];

 HighlightGraph[envyGraph[],
      step3Styles[e, d["indifferent upstream poor agent edges"],
       d["indifferent downstream rich agent edges"]]] //
     addLegends[#, step3Legend] & //
    Labeled[#, str["Before sidepayment. With object allocation σ=",
       σ, ", we have a strong edge ", e, "."], Top] & // Framed // Print;

 sidepaymentProcedure[e];

 HighlightGraph[envyGraph[], {styleEdge[d["new weak edge"], Red]}] //
    Labeled[#,
      str["After sidepayment: Moving $", N@d["least money for new weak edge"],
       "from rich=", d["indifferent downstream rich agents"], "to poor=",
       d["indifferent upstream poor agents"], "creates new weak edge",
       d["new weak edge"], "."], Top] & // Framed // Print;
]
```


Before sidepayment. With object allocation
σ= {5, 4, 3, 2, 1} , we have a strong edge 1 ↦ 4 .
1
5
$0.
5
1
$0.
2
4
$0.
4
2
$0.
3
3
$0.
envious agent
not envious agent
envies
indifferent to
∑Utility=31100.
strong edge
indifferent upstreams
indifferent downstreams



Final Algorithm

Algorithm: Let a feasible allocation (σ, x) be given. Consider its envy-graph.

Step 1. If there are no strong edges, then we have an envy-free allocation. Stop. Otherwise go to step 2.

Step 2. If there is a cycle containing a strong edge, then apply the **permutation procedure** and go to step 1. Otherwise pick a strong edge and go to step 3.

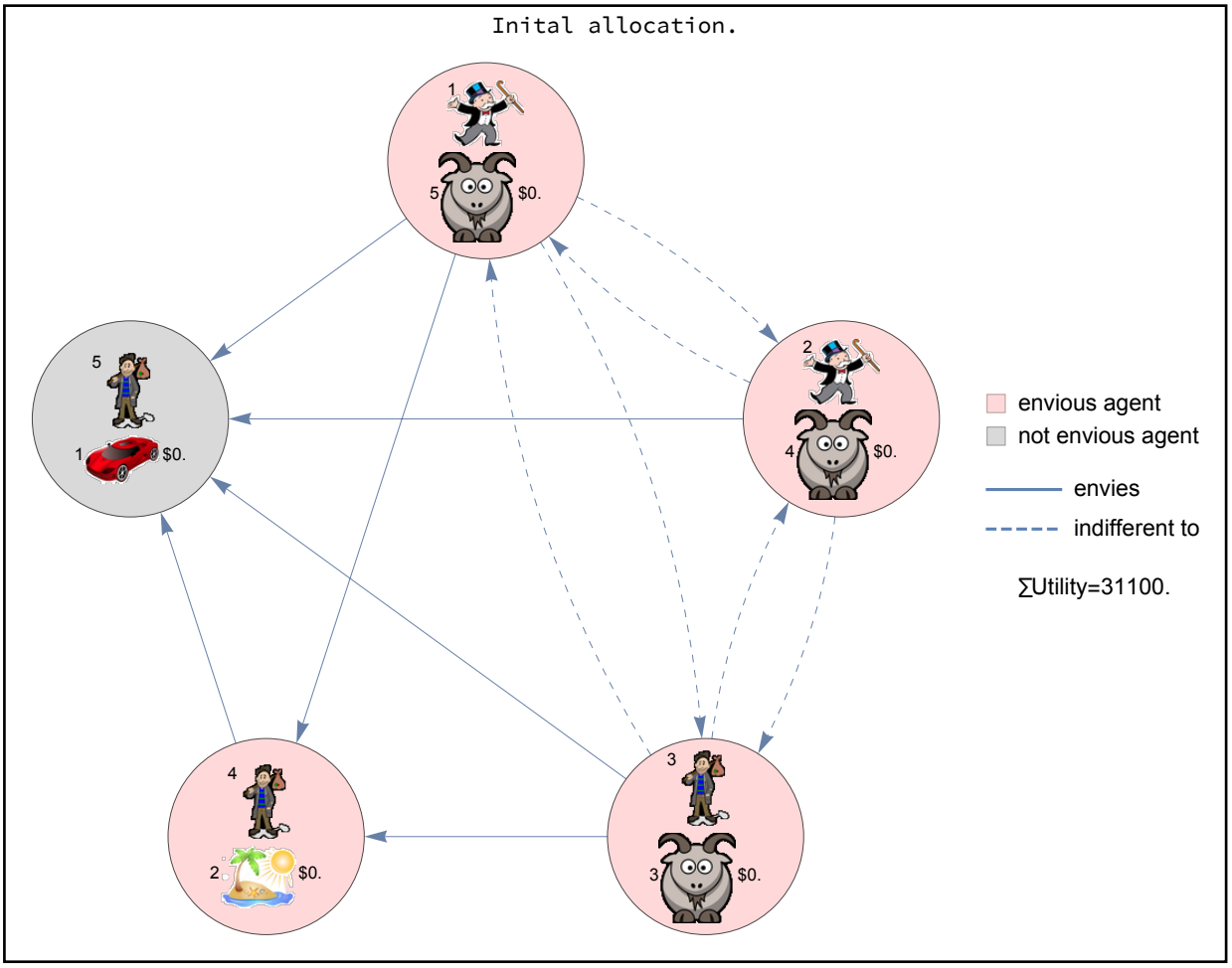
Step 3. Apply the **sidepayment procedure** to the picked edge and go to step 4.

Step 4. If a strong edge has been eliminated, go to step 1. If a cycle with a strong edge has emerged, go to step 2. Otherwise go to step 3.

In[54]:=
```
initializeAllocation
nsteps = 0; maxsteps = 100;
goto[lab_] := If[nsteps++ < maxsteps, Goto[lab],
   Print@str["MAX GOTOS (", nsteps, ") EXCEEDED, BAILING!"];
   Goto[end]];
savePic[g_] := (Export["pics/envy-free-" <> IntegerString[nsteps, 10, 3] <> ".png",
```

```
   g, ImageSize → 500 {16 / 9, 1}];
  g)

Module[{},
 Print["BEGIN:"];
 envyGraph[] // addLegends // Labeled[#, "Inital allocation.", Top] & // Framed //
   savePic // Print;

 Label[step1];
 Print["Start Step 1: If no strong edges, done. Else Step 2."];
 If[strongEdges == {},
  goto[end],
  goto[step2]
 ];

 Label[step2];
 Print["Start Step 2: If ∃ cycle w/ strong edge,
    permute then Step 1. Else pick strong edge then Step 3."];
 If[hasCycleWithStrongEdge,
  c = getCycleWithStrongEdge;
  envyGraph[] // HighlightGraph[#, styleEdge@c] & // addLegends //
      Labeled[#, str["Found cycle with strong edge", c, ":"], Top] & //
     Framed // savePic // Print;
  permutationProcedure[c];
  goto[step1],
  (* else *)
  SeedRandom[123]; e = RandomChoice@strongEdges;
  Print@str["Picked strong edge", e];
  goto[step3]
 ];

 Label[step3];
 Print["Start Step 3: sidepayment then Step 4."];
 numOfStrongEdgesPre3 = Length@strongEdges;
 d = sidepaymentData[e];
 HighlightGraph[envyGraph[],
        step3Styles[e, d["indifferent upstream poor agent edges"],
         d["indifferent downstream rich agent edges"]]] //
       addLegends[#, step3Legend] & //
      Labeled[#, str["Before Sidepayment: We have a strong edge ", e, "."],
        Top] & // Framed // savePic // Print;
 sidepaymentProcedure[e];
 Print@str["Moving $", N@d["least money for new weak edge"],
```

```
    "from rich=", d["indifferent downstream rich agents"],
    "to poor=", d["indifferent upstream poor agents"],
    "creates new weak edge", d["new weak edge"], "."];
  goto[step4];

  Label[step4];
  Print["Start Step 4: If # strong edges ↓, Step
     1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3."];
  Which[
   Length@strongEdges < numOfStrongEdgesPre3, goto[step1],
   hasCycleWithStrongEdge, goto[step2],
   True, goto[step3]
  ];

  Label[end];
  Speak["Done!"];
  Print["Done! Should be no strong edges:"];
  envyGraph[] // addLegends // Labeled[#, "Final allocation.", Top] & // Framed //
    savePic // Print;

]
BEGIN:
```

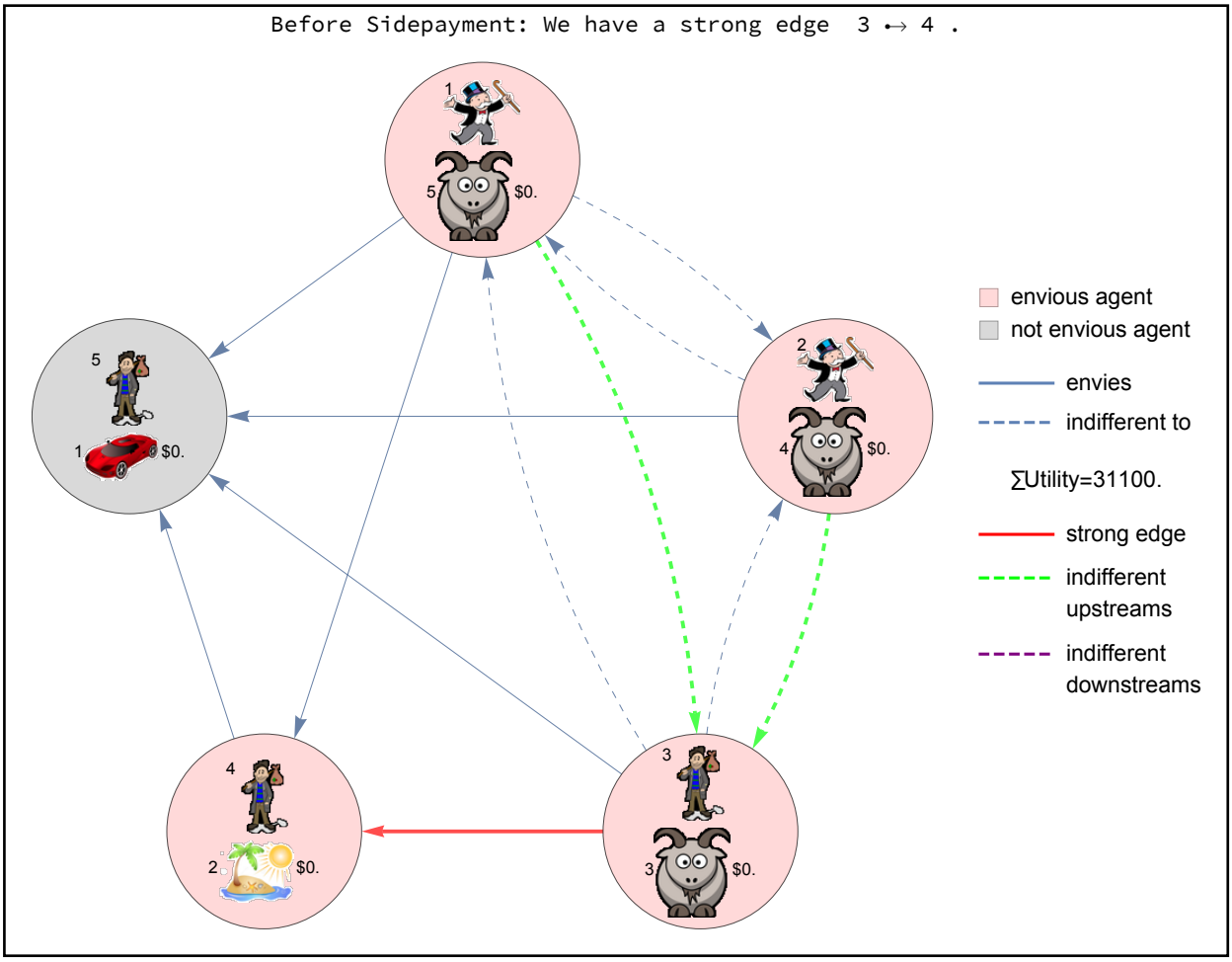


Start Step 1: If no strong edges, done. Else Step 2.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.

Picked strong edge 3 ⇢ 4

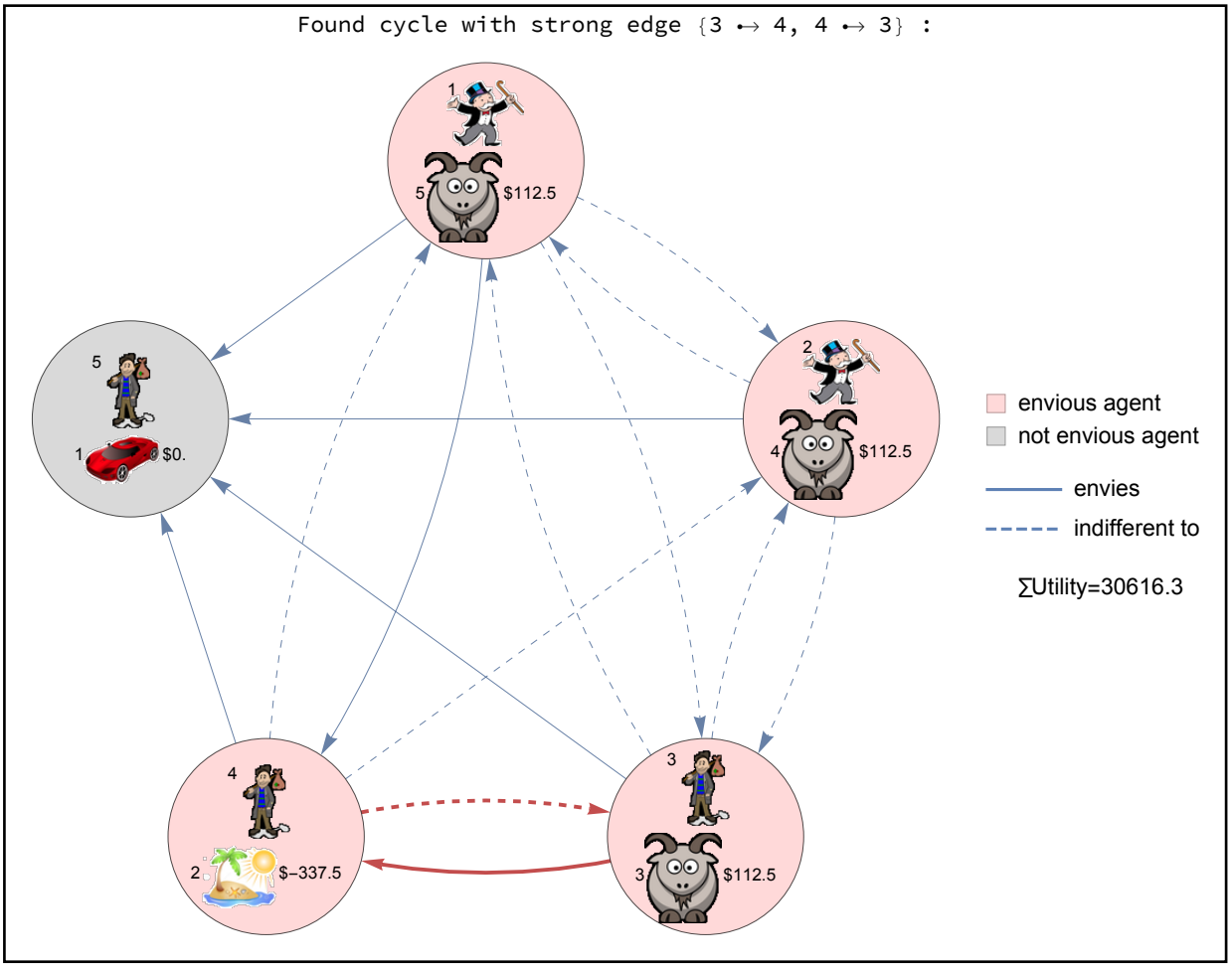
Start Step 3: sidepayment then Step 4.



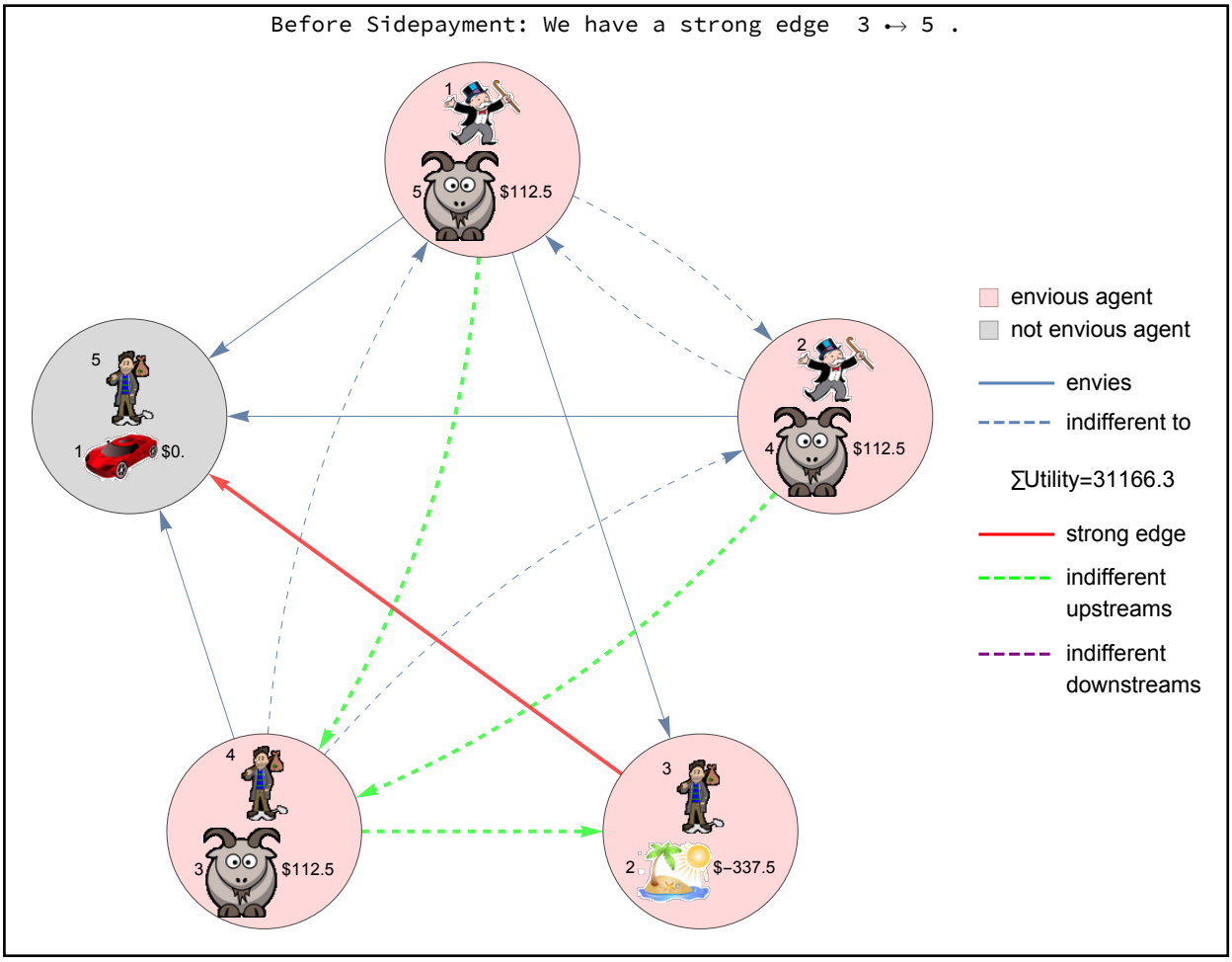
Moving $ 337.5 from rich= {4} to poor= {3, 1, 2} creates new weak edge 4 ⇢ 1 .

Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.



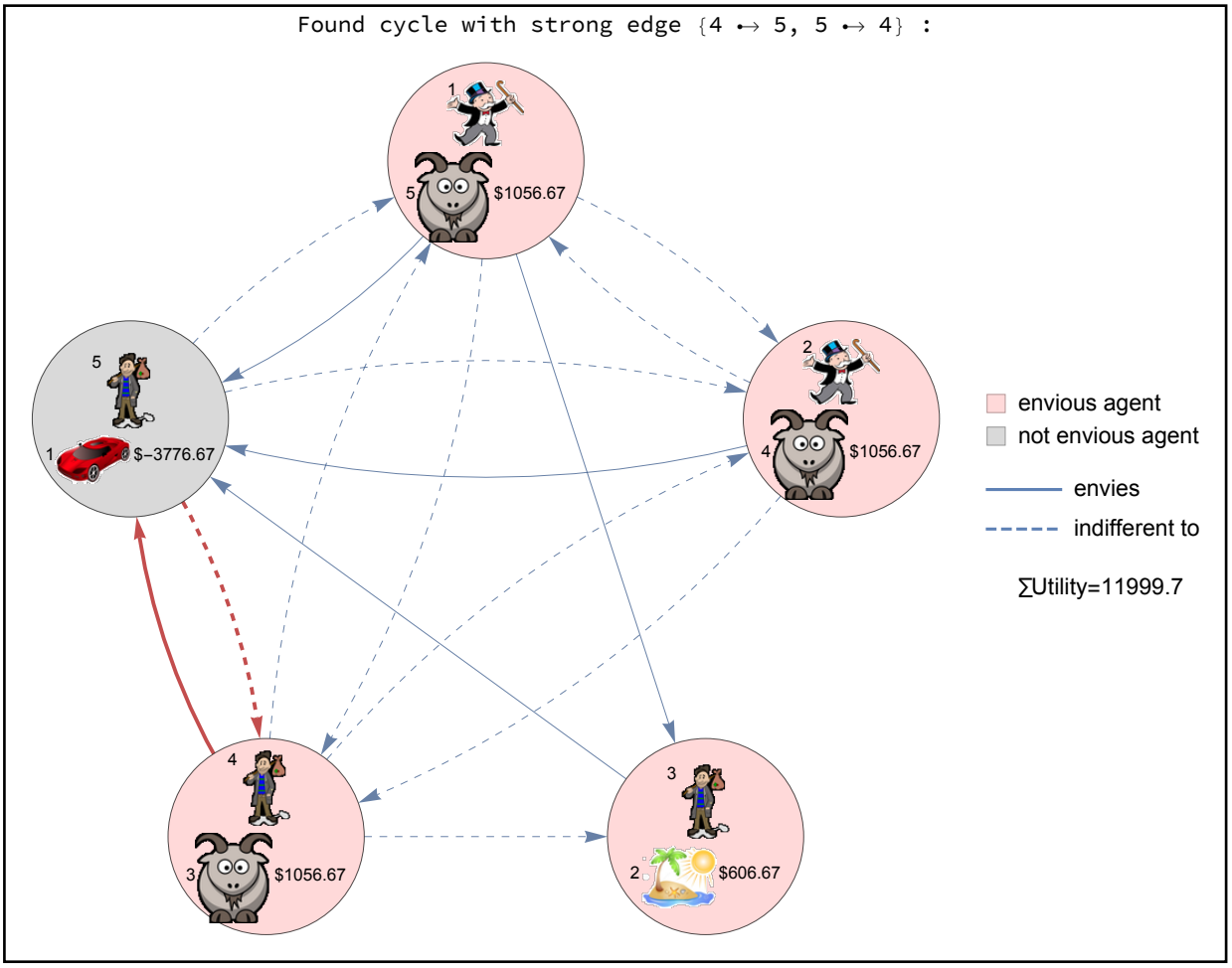

```
Permuting...

Start Step 1: If no strong edges, done. Else Step 2.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.

Picked strong edge 3 ⇢ 5

Start Step 3: sidepayment then Step 4.
```



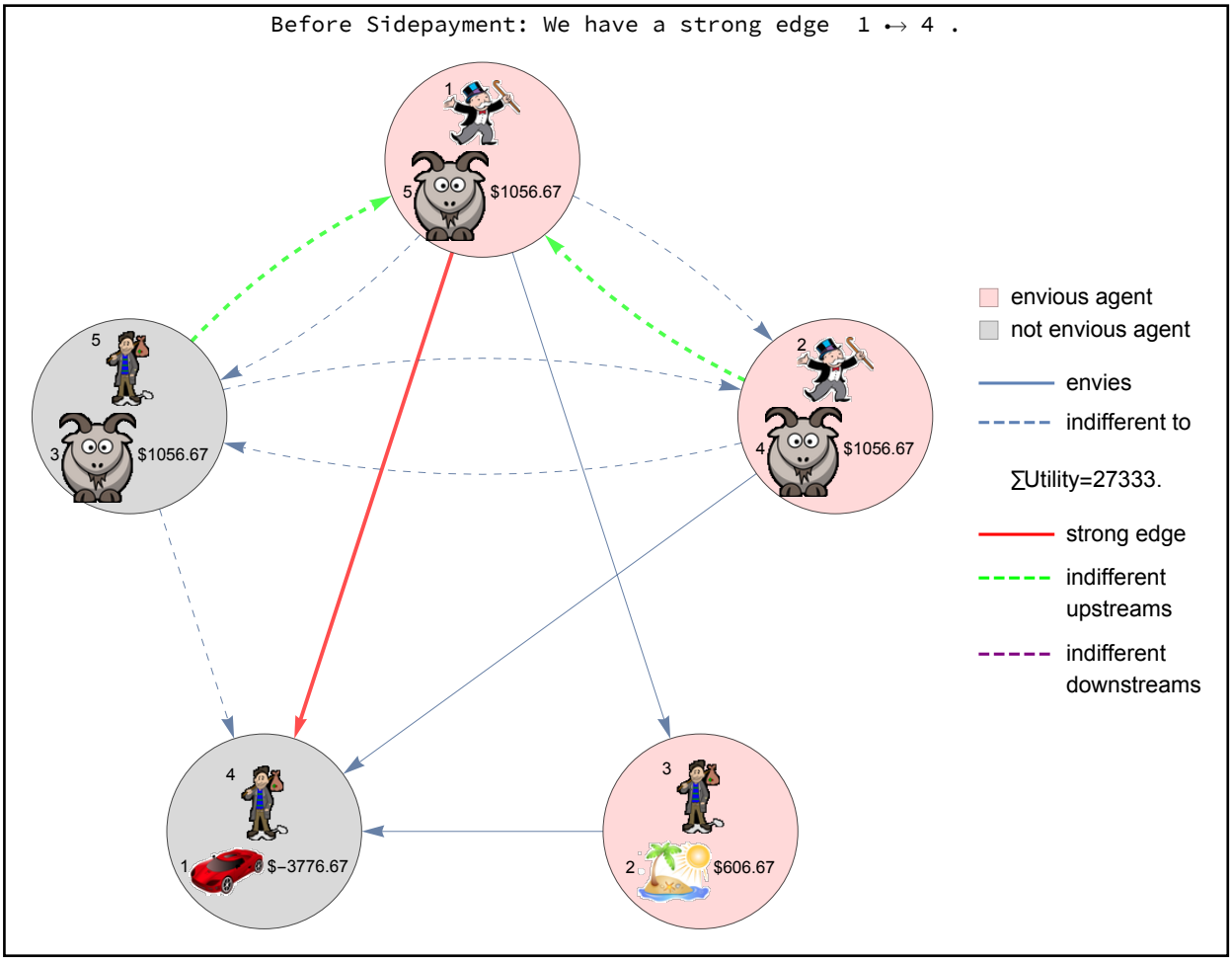
Moving $ 3776.67 from rich= {5} to poor= {3, 4, 1, 2} creates new weak edge 5 ⇢ 1 .

Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.




```
Permuting...

Start Step 1: If no strong edges, done. Else Step 2.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.

Picked strong edge 1 ⇢ 4

Start Step 3: sidepayment then Step 4.
```



Moving $ 150. from rich= {4} to poor= {1, 2, 5} creates new weak edge 1 ⇢ 3 .

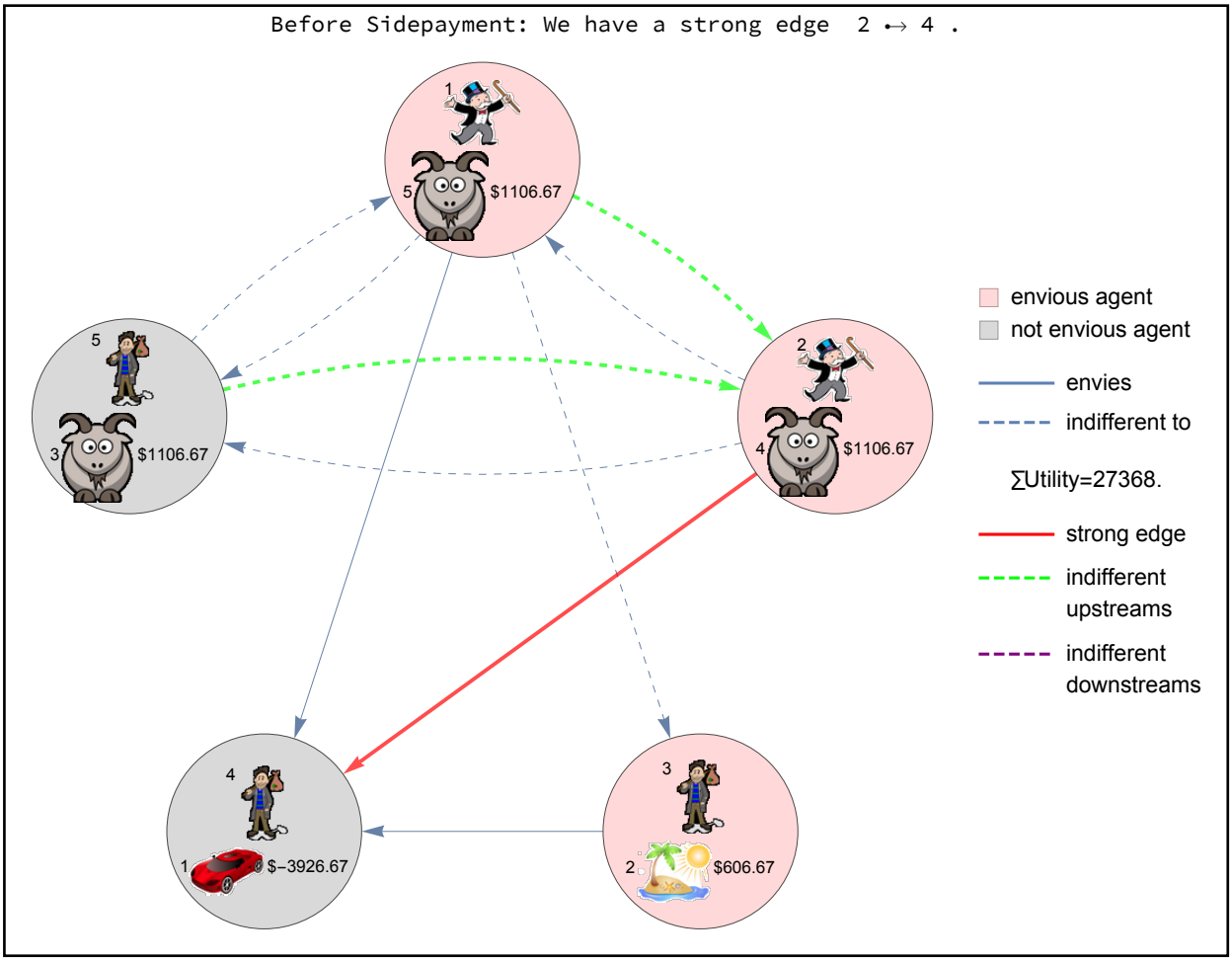
Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

Start Step 1: If no strong edges, done. Else Step 2.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.

Picked strong edge 2 ⇢ 4

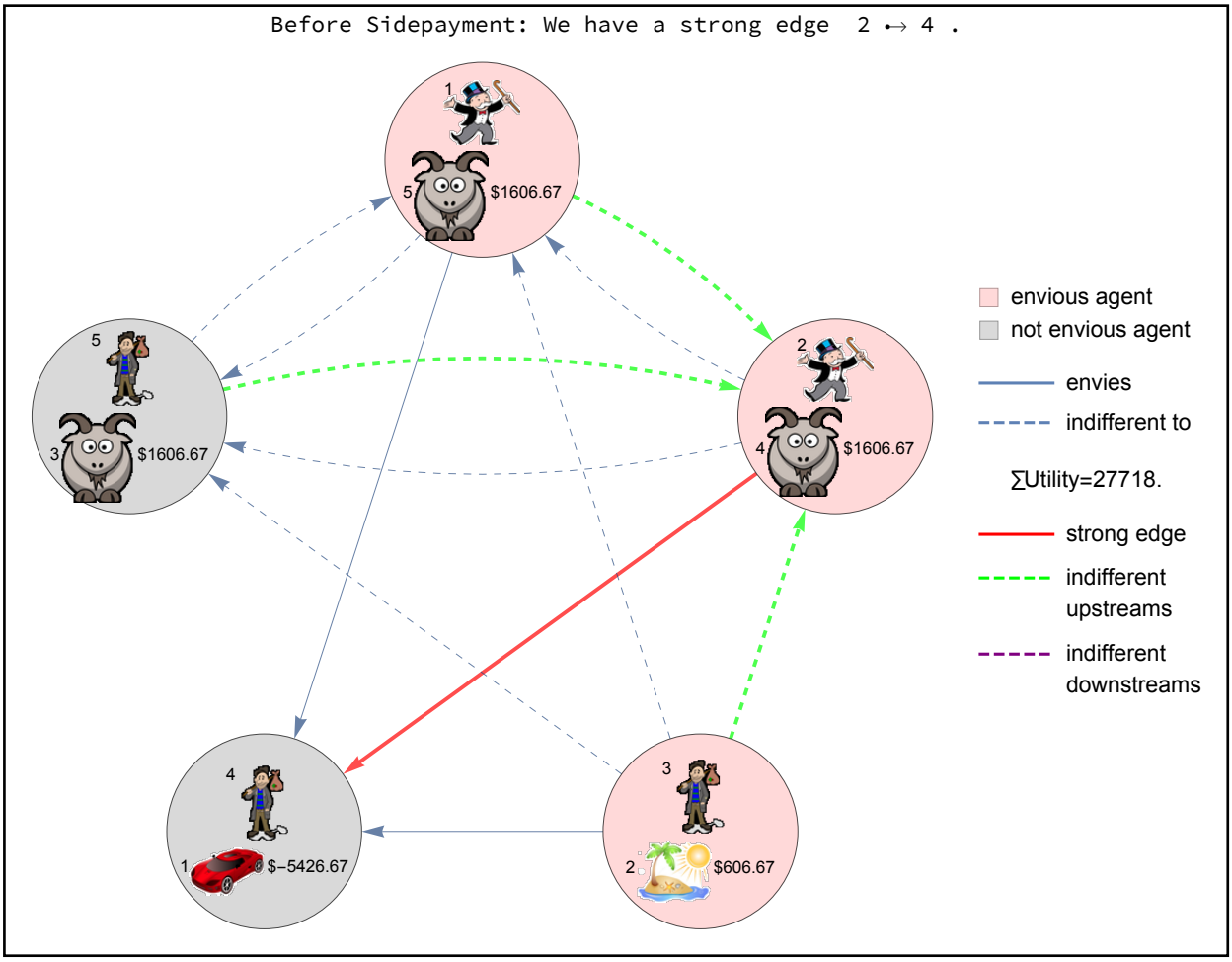
Start Step 3: sidepayment then Step 4.



Moving $ 1500. from rich= {4} to poor= {2, 1, 5} creates new weak edge 3 ⇢ 1 .

Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

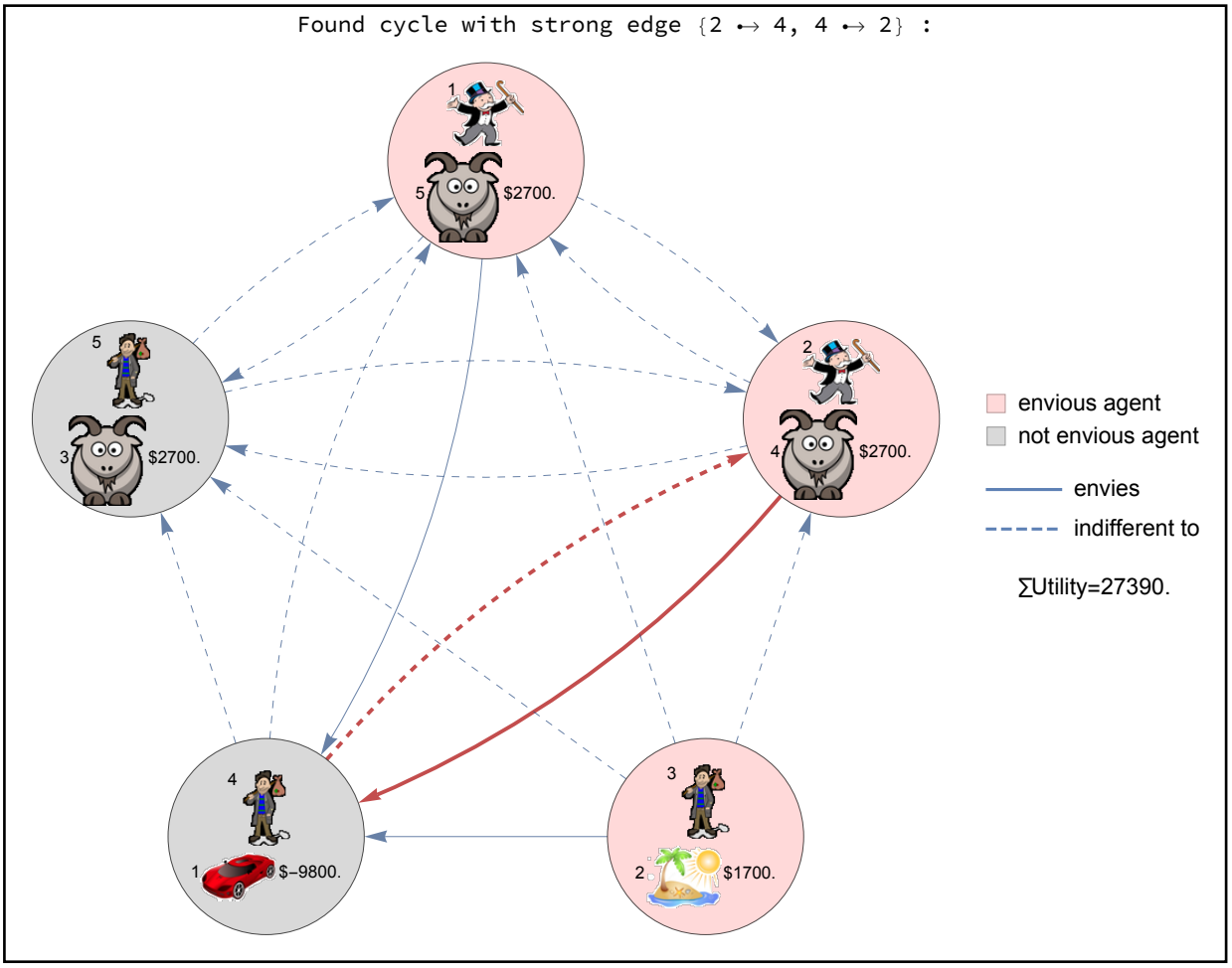
Start Step 3: sidepayment then Step 4.



Moving $ 4373.33 from rich= {4} to poor= {2, 1, 3, 5} creates new weak edge 4 ⇢ 1 .

Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.



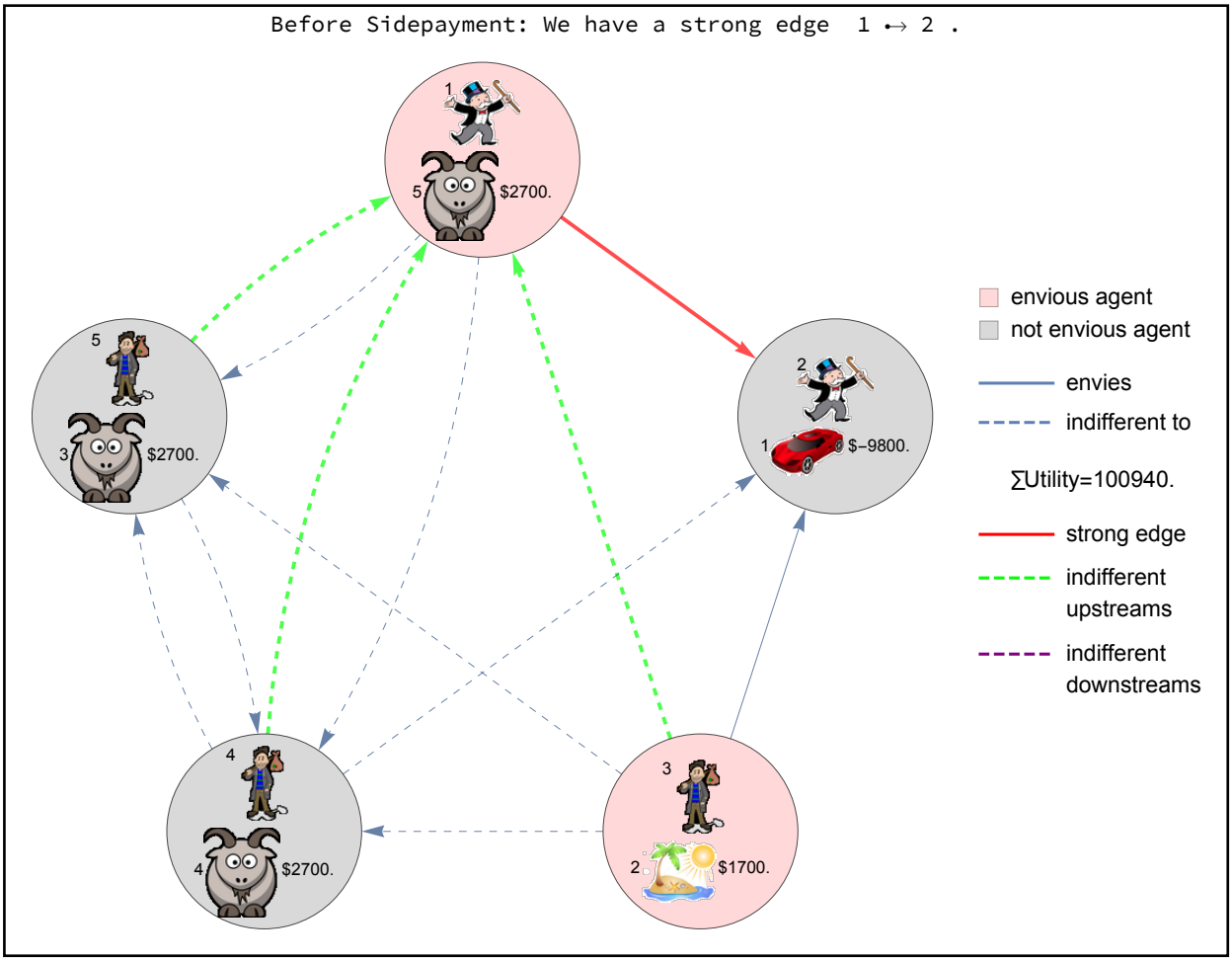
Permuting...

Start Step 1: If no strong edges, done. Else Step 2.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.

Picked strong edge 1 ⇢ 2

Start Step 3: sidepayment then Step 4.



Moving $ 6000. from rich= {2} to poor= {1, 3, 4, 5} creates new weak edge 3 ⇢ 2 .

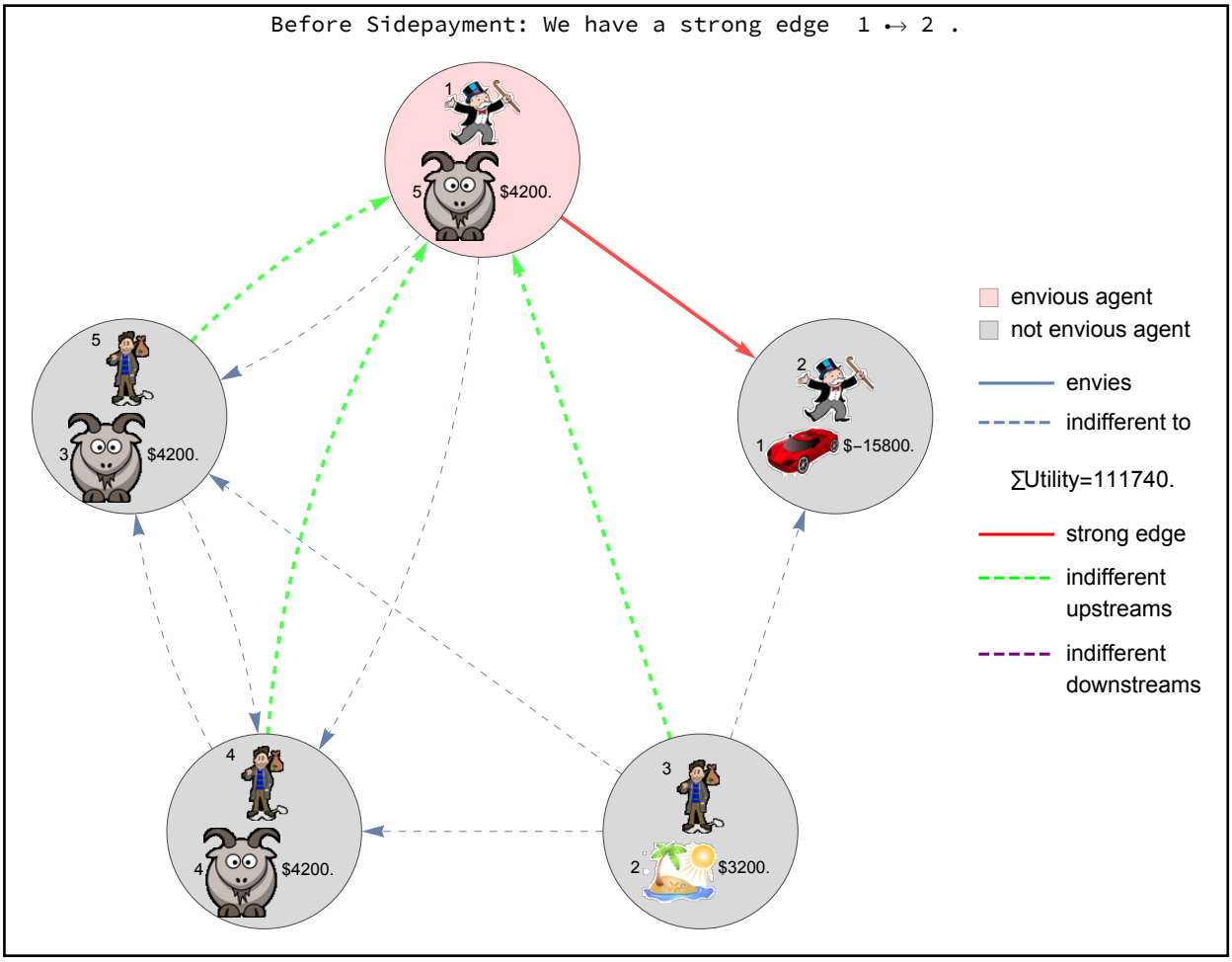
Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

Start Step 1: If no strong edges, done. Else Step 2.

Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.

Picked strong edge 1 ⇢ 2

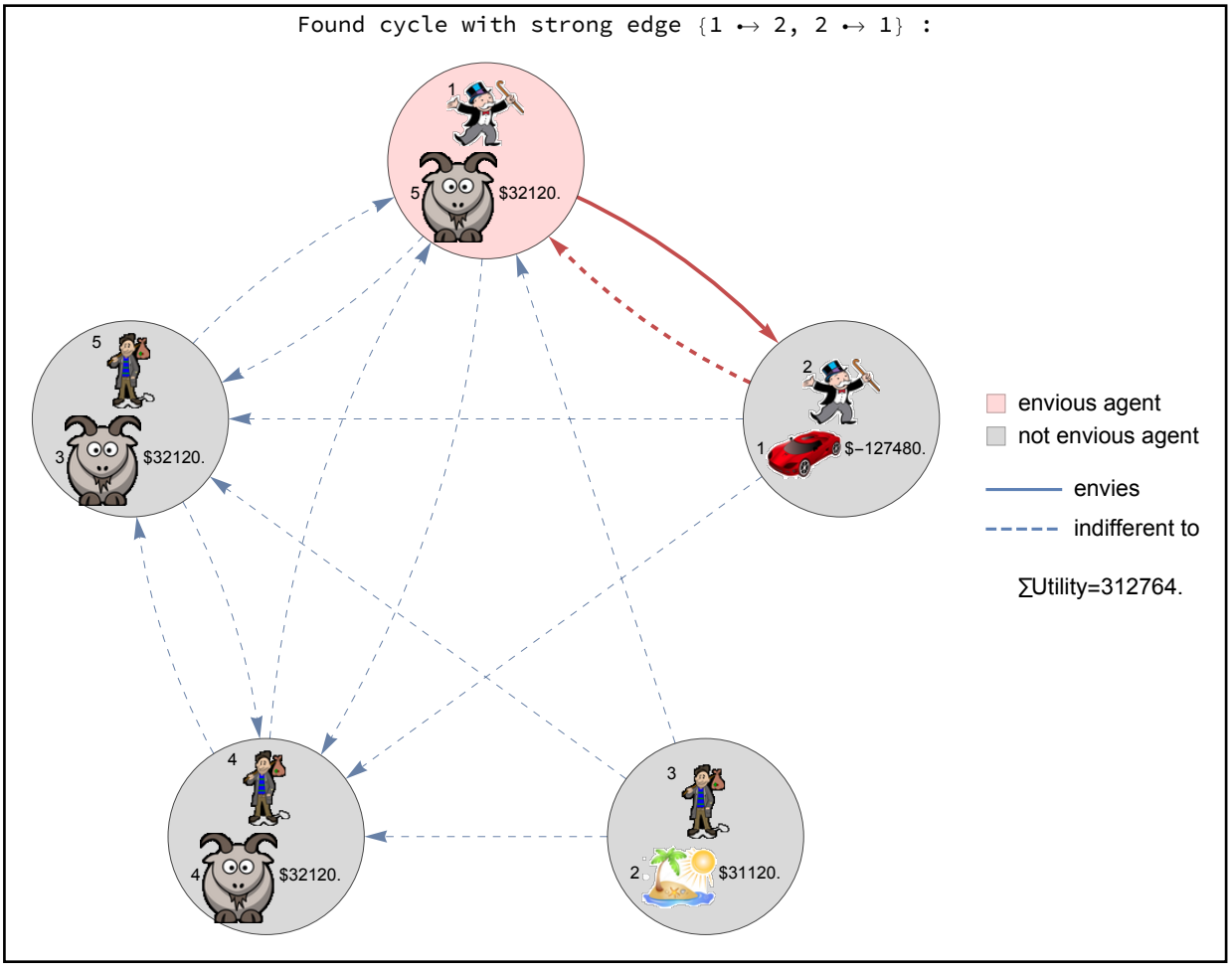
Start Step 3: sidepayment then Step 4.



Moving $ 111680. from rich= {2} to poor= {1, 3, 4, 5} creates new weak edge 2 ⇢ 1 .

Start Step 4: If ⌗ strong edges ↓, Step
  1. Else if ∃ cycle w/ strong edge, Step 2. Else Step 3.

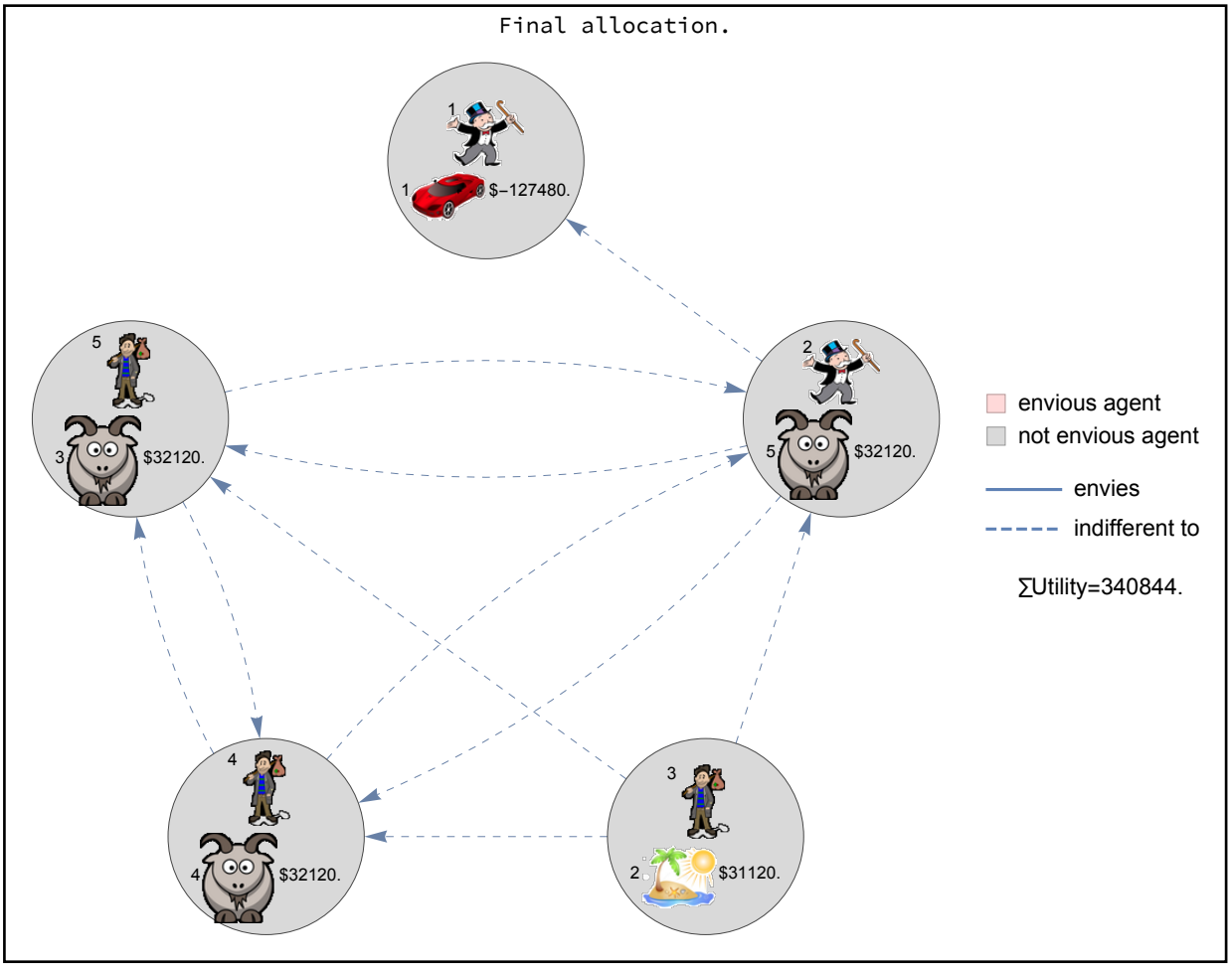
Start Step 2: If ∃ cycle w/ strong edge,
  permute then Step 1. Else pick strong edge then Step 3.



Permuting...

Start Step 1: If no strong edges, done. Else Step 2.

Done! Should be no strong edges:



No net money is required; the agents pay each other:

In[59]:=
```
Total[x]
```

Out[59]=
```
0
```

Here are the final object allocations and money allocations:

In[60]:=
```
showAllocationTable["Final object and money allocations."]
```

Out[60]=

Final object and money allocations.

agent	object	money	utility
1	1	−127 480	34 504
2	5	32 120	16 260
3	2	31 120	32 120
4	4	32 120	64 240
5	3	32 120	193 720

Here is the final envy table. No one envies anyone else!

In[61]:=
```
showEnvyTable["Final envy table. (Should be no envy.)"]
```

Final envy table. (Should be no envy.)

0	−28 080	−28 180	−28 080	−28 080
0	0	−630	0	0
−139 600	0	0	0	0
−294 200	0	−1100	0	0
−928 600	0	−5900	0	0

envy > 0

Save animation

In[62]:=
```
FileNames["pics/*.png"] // Map[Import] //
  Export["envy-free.gif", #, "AnimationRepetitions" → ∞, "DisplayDurations" → .5] &;
```

In[63]:=