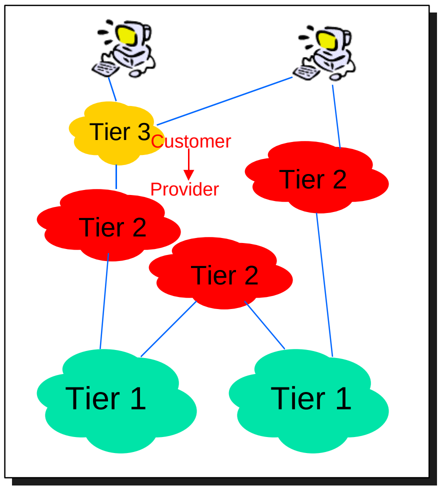
Inter-Domain Routing

Arvind Krishnamurthy

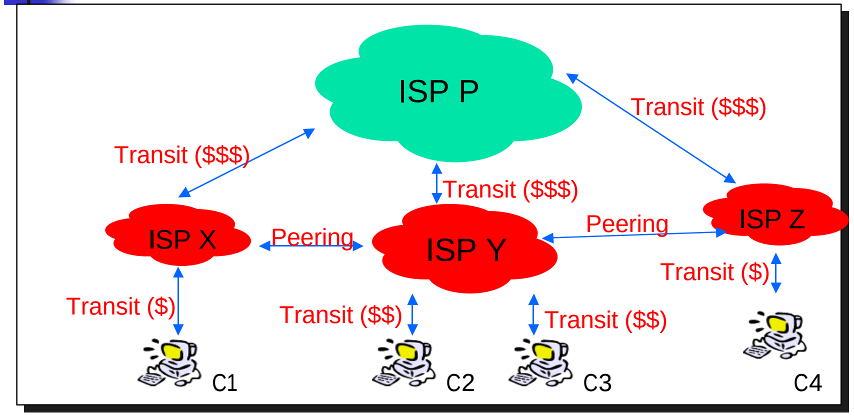
Fall 2003

A Logical View of the Internet

- Hierarchical ordering of autonomous systems
  - Area routing
    - Details of internals not necessary for routing
  - Provides scalability but might degrade performance
- Relationships between autonomous systems:
  - Transit relationship: paying relationship
  - Peering relationship: non-paying one
    - Avoid traffic through a transit relationship
    - Find lower latency paths

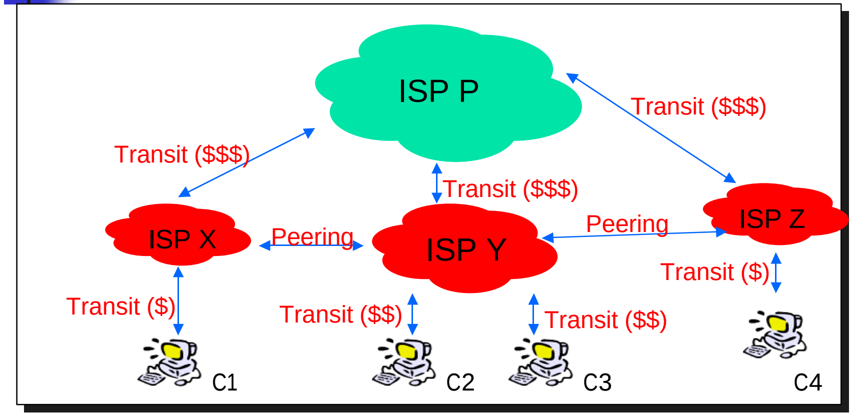


Economics of the Internet



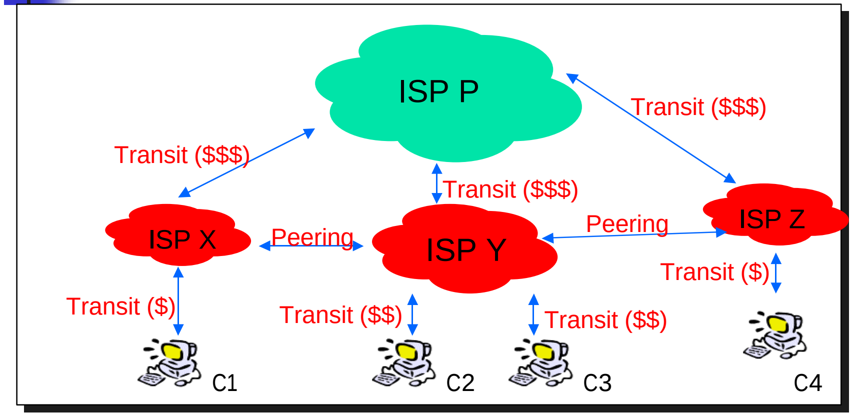
- Transit relationship:
  - Reveal customers to everyone
  - Reveal all known paths to customers
  - ISP Y will advertise paths to C2 and C3

Peering Relationships



- ISP Y peers with ISP Z
  - Y informs Z of C2 and C3
  - Z informs Y of C4

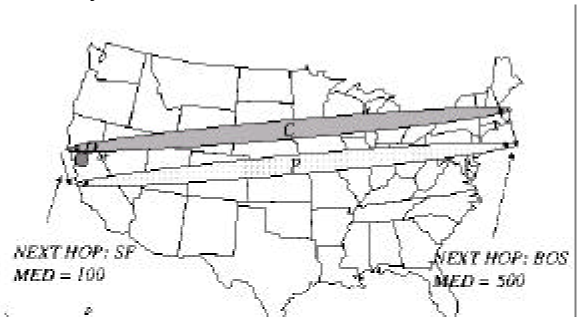
Peering Relationships



- ISP Y peers with ISP X
  - Y informs X of C2 and C3
  - X informs Y of C1
  - Y does not tell X of the path to C4
  - Y needs to have transit relationship with P to route to C4

Multi-Exit Discriminator

- C has a transit relationship with P
- C needs to send a packet into P's domain
  - Packet origin: Boston, packet destination: SF
  - It has two choices: transit into P earlier (near Boston) or transit later (near SF)
  - Prefer early transit



Routing Preferences

- In general, customer > peer > provider
  - Use LOCAL PREF to ensure this
  - Routing through customer and peer could lower latency and lower traffic costs
- Processing order of path attributes:
  - Select route with highest LOCAL-PREF
  - Select route with shortest AS-PATH
  - For routes learned from same neighbor:
    - Apply multi-exit discriminator

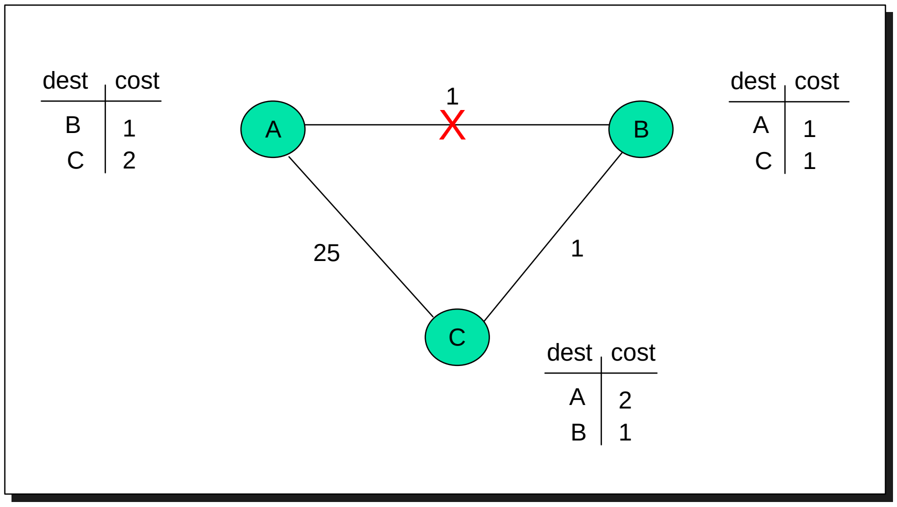
Routing algorithms for the internet

- Link state or distance vector?
- Problems with link state:
  - LS database too large – entire Internet
  - Metric used by routers not the same
  - May expose policies to other AS's
- Can we use distance-vector algorithms for policy routing?

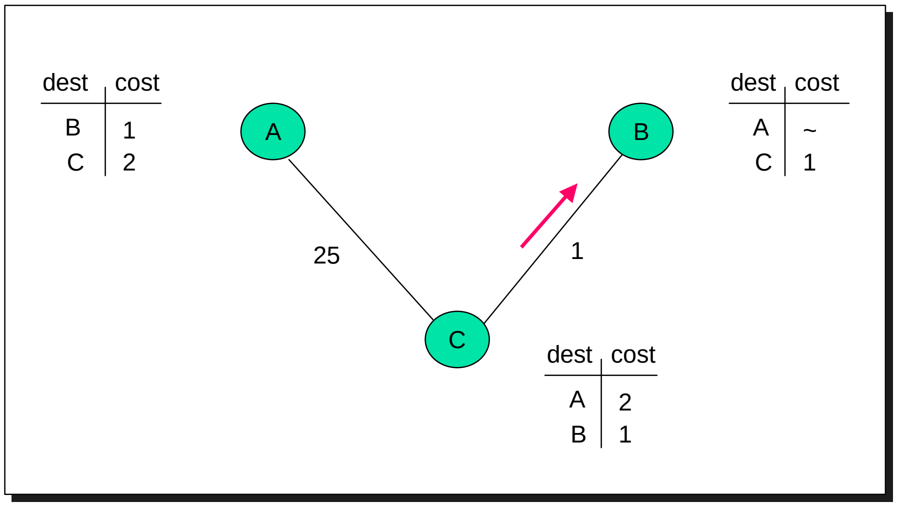
Announcements

- Reminder:
  - First quiz will be held next Monday
- Project dynamics will be announced later this week

The Bouncing Effect



C Sends Routes to B

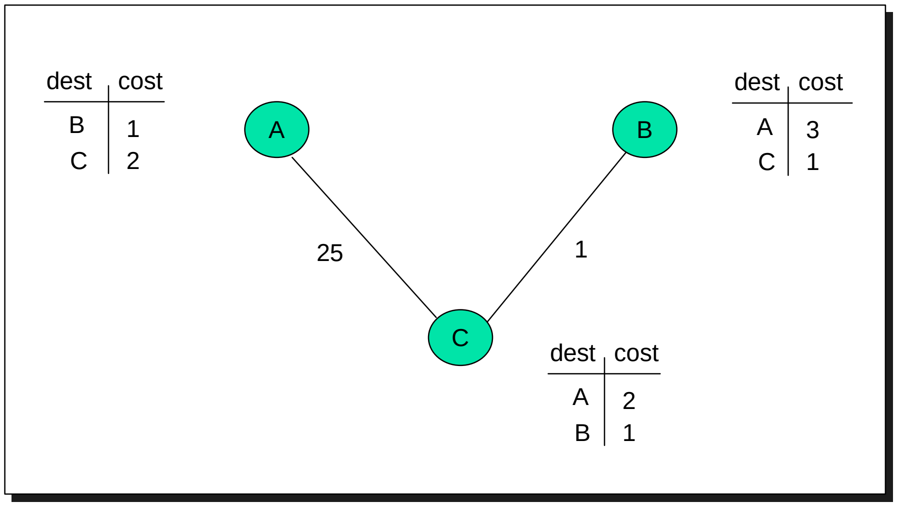


dest	cost
B	1
C	2

dest	cost
A	~
C	1

dest	cost
A	2
B	1

B Updates Distance to A

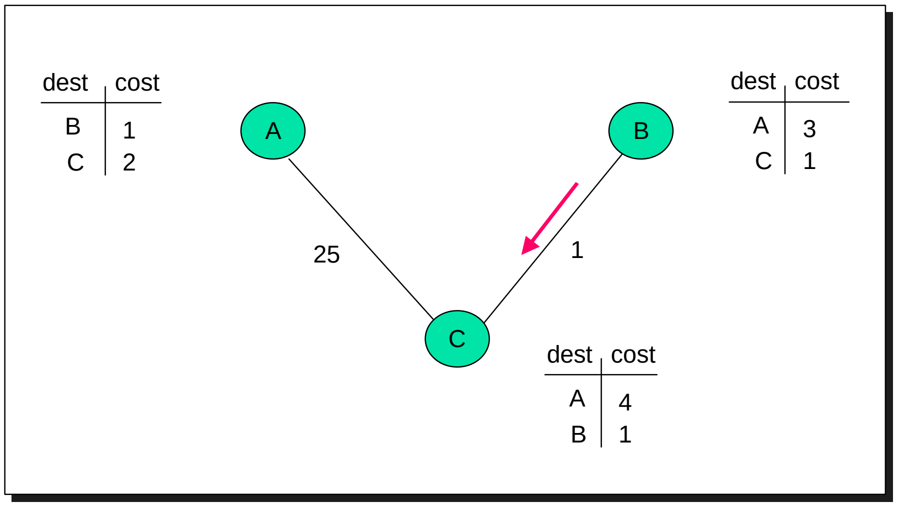


dest	cost
B	1
C	2

dest	cost
A	3
C	1

dest	cost
A	2
B	1

B Sends Routes to C

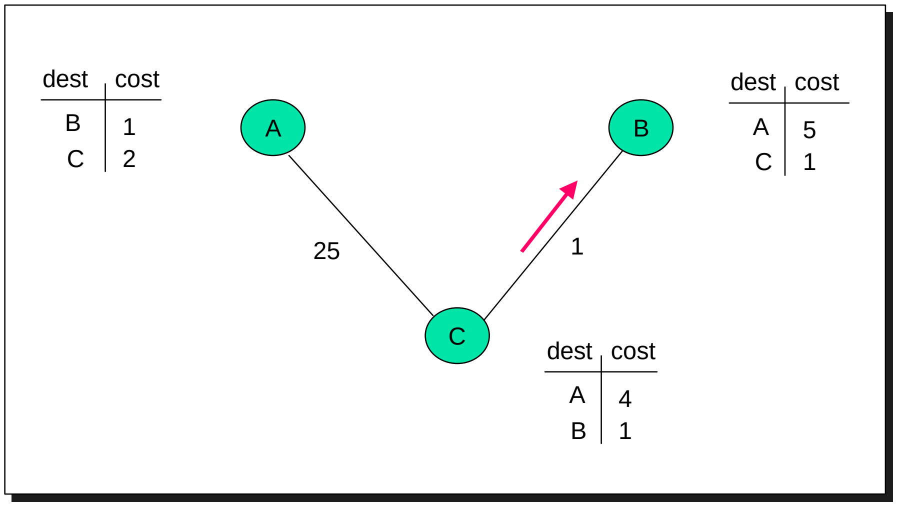


dest	cost
B	1
C	2

dest	cost
A	3
C	1

dest	cost
A	4
B	1

C Sends Routes to B



dest	cost
B	1
C	2

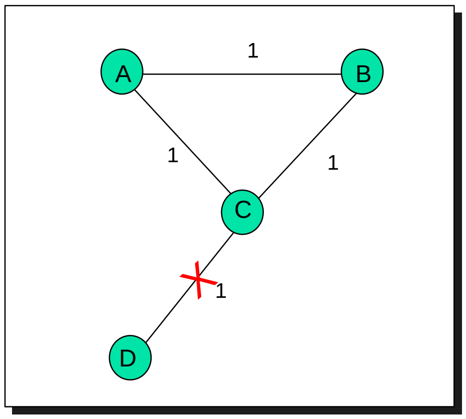
dest	cost
A	5
C	1

dest	cost
A	4
B	1

Solutions

- Problems arise:
  - When metric increases
  - Implicit path has loops
- Solution 1: If metric increases, delay propagating information
  - In our example, B delays advertising route
  - C eventually thinks B's route is gone, picks its own route
  - B then selects C as next hop
- Adversely affects convergence
- Other "Solutions":
  - Split horizon: C does not advertise route to B
  - Poisoned reverse: C advertises route to B with infinite distance
- Works for two node loops
  - Does not work for loops with more nodes

Example Where Split Horizon Fails



- When link breaks, C marks D as unreachable and reports that to A and B
- Suppose A learns it first
  - A now thinks best path to D is through B
  - A reports D unreachable to B and a route of cost=3 to C
- C thinks D is reachable through A at cost 4 and reports that to B
- B reports a cost 5 to A who reports new cost to C
- etc...

Solution: Distance Vector with Path

- Each routing update carries the entire path
- Loops are detected as follows:
  - When AS gets route check if AS already in path
    - If yes, reject route
    - If no, add self and (possibly) advertise route further
- Advantage:
  - Metrics are local - AS chooses path, protocol ensures no loops
- Hop-by-hop Model
  - BGP advertises to neighbors only those routes that it uses
    - Consistent with the hop-by-hop Internet paradigm
    - e.g., AS1 cannot tell AS2 to route to other AS's in a manner different than what AS2 has chosen (need source routing for that)

What problem is BGP solving?

Underlying problem	Distributed means of computing a solution.
Shortest Paths	BF: RIP, OSPF …
X?	BGP

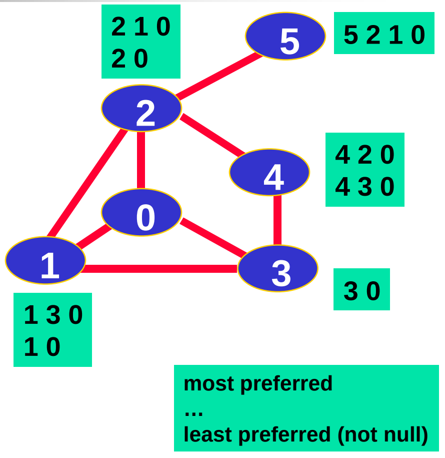
Having an X can

- **aid in the design of policy analysis algorithms and heuristics,**
- **aid in the analysis and design of BGP and extensions,**
- **help explain some BGP routing anomalies,**
- **provide a fun way of thinking about the protocol**

Stable-Paths Problem

Input:

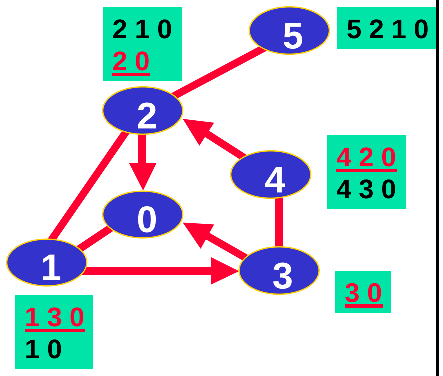
- A graph of nodes and edges,
- Node 0, called *the origin*,
- For each non-zero node, a set or permitted paths to the origin. This set always contains the “null path”.
- A ranking of permitted paths at each node. Null path is always least preferred. (Not shown in diagram)



Solution to SPP

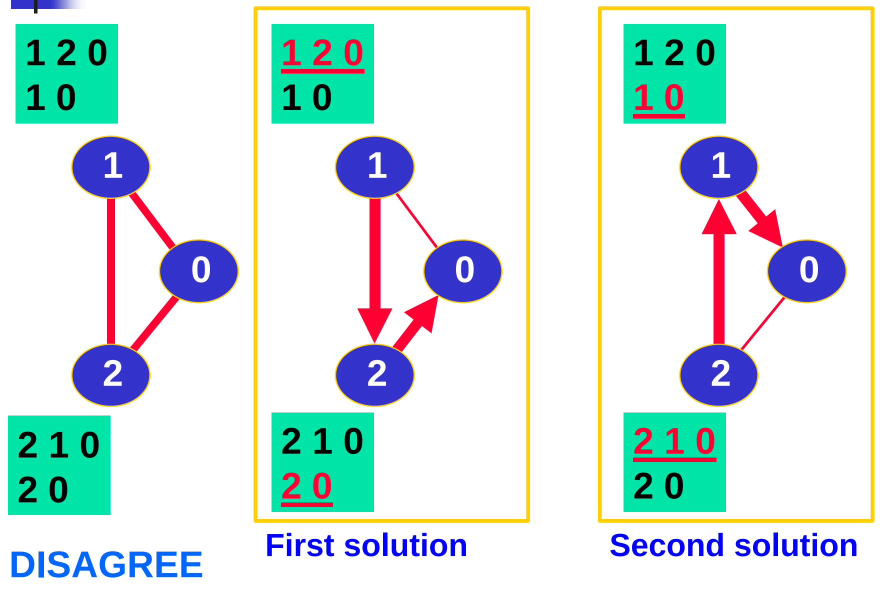
A *solution* is an assignment of permitted paths to each node such that

- **node u’s assigned path is either the null path or is a path uwP, where wP is assigned to node w and {u,w} is an edge in the graph,**
- **each node is assigned the highest ranked path among those consistent with the paths assigned to its neighbors.**

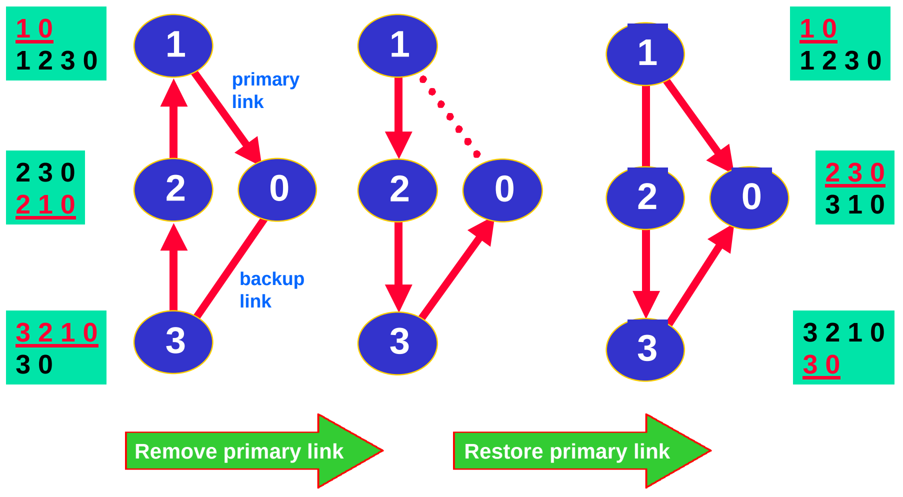


A Solution need not represent a shortest path tree, or a spanning tree.

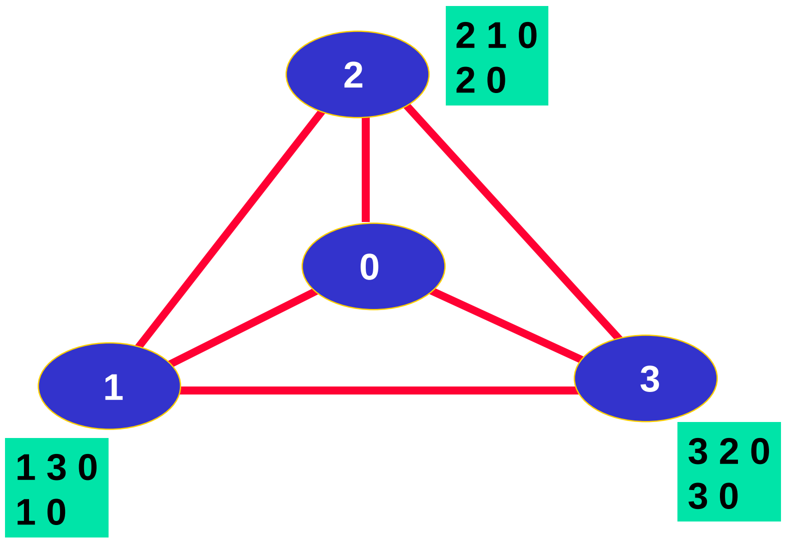
SPP may have multiple solutions



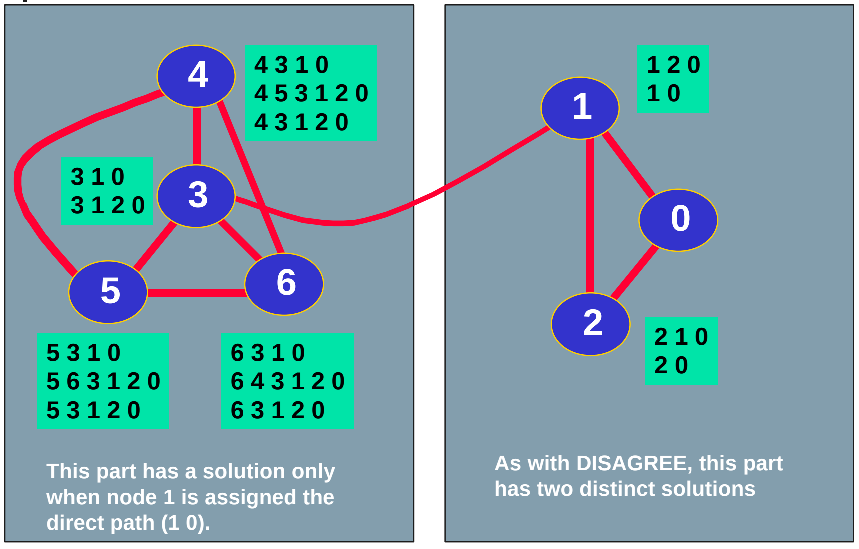
“Route Triggering”



BAD GADGET: No solution



PRECARIOUS: Has solution, but can get trapped



This part has a solution only when node 1 is assigned the direct path (1 0).

As with DISAGREE, this part has two distinct solutions

Solving SPP

Just enumerate all path assignments
And check stability of each….

Exponential complexity !!

But, in worst case you (probably)
can't do any better…

3-SAT

Variables V = {X1, X2, …, Xn}

Clauses C1 = X17 or ~X23 or ~X3,
C2 = ~X2 or X3 or ~X12
….
Cm = X6 or ~X7 or X18

Question Is there an variable assignment
A : V $\rightarrow$ {true, false} such that
each clause C1, … ,Cm is true?

3-SAT is NP-complete

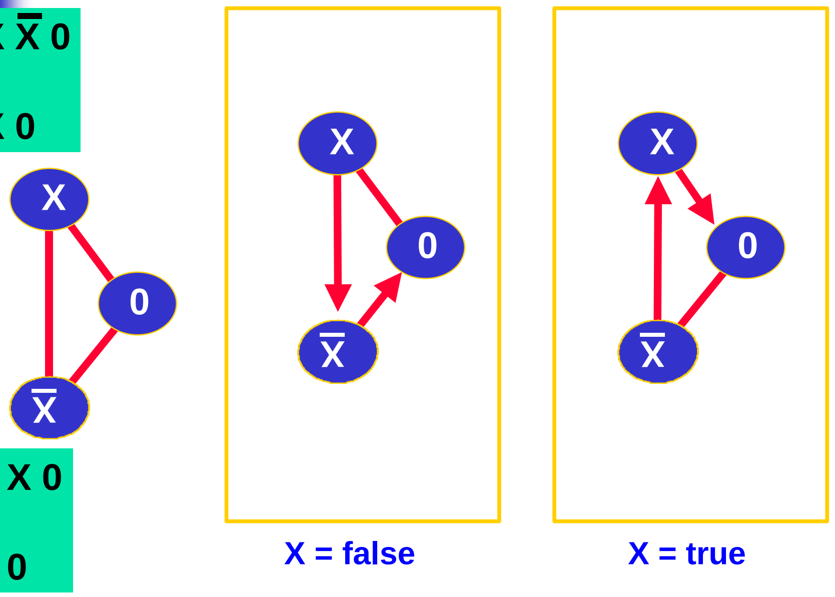
Assignment to variable X

X $\bar{X}$ 0

X 0

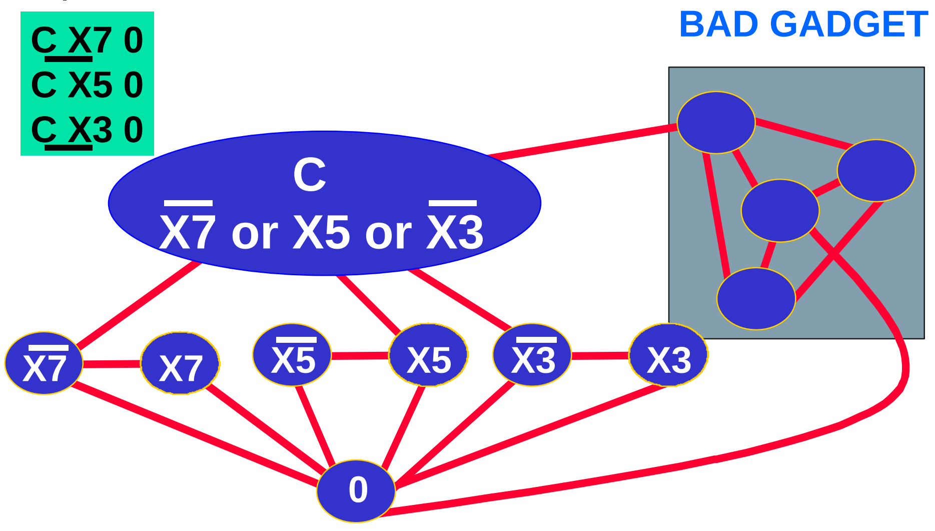
$\bar{X}$ X 0

$\bar{X}$ 0



Solvability is NP-Complete

C $\underline{X7}$ 0
C X5 0
C $\underline{X3}$ 0



Simple Path Vector Protocol (SPVP)

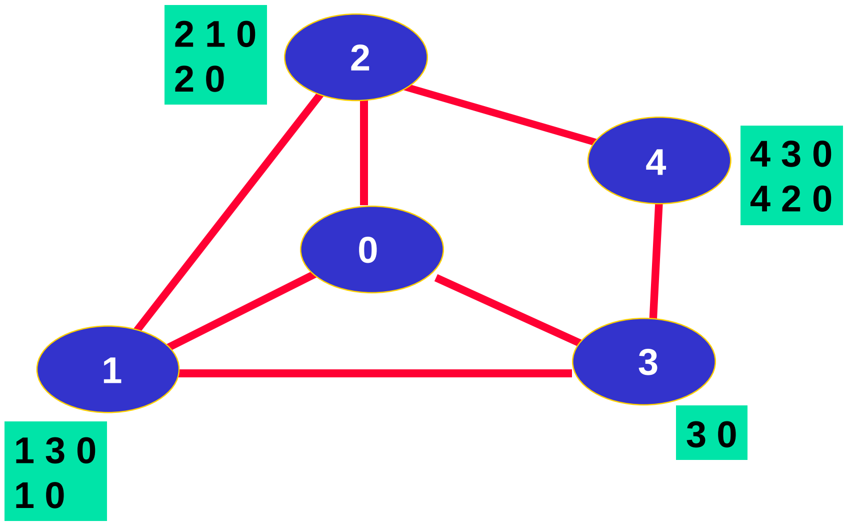
Pick the best path available at any given time…

rib(u): u's best path to origin

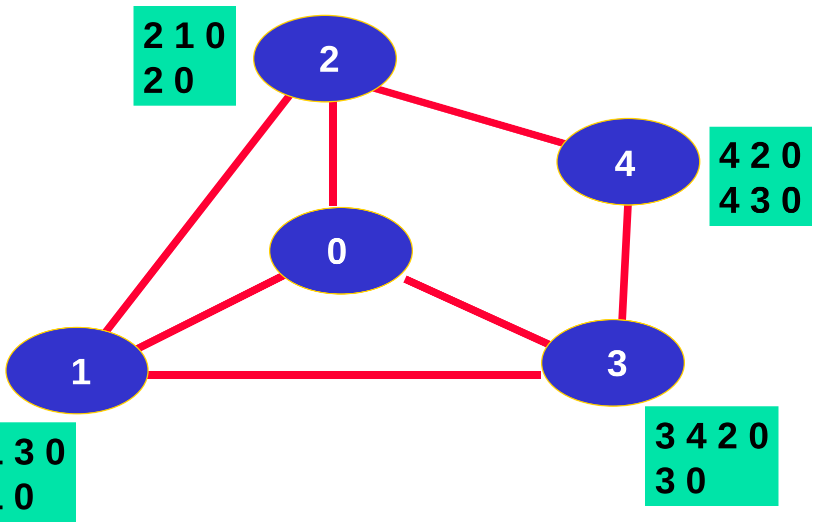
rib-in(u←w): recent path sent from w

```
process spvp[u]
{
    receive P from w →
    {  rib-in(u←w) := u P
       if rib(u) != best(rib-in(u←x)) {
          rib(u) := best(rib-in(u←x))
          foreach v in peers(u) {
             send rib(u) to v
          }
       }
    }
}
```

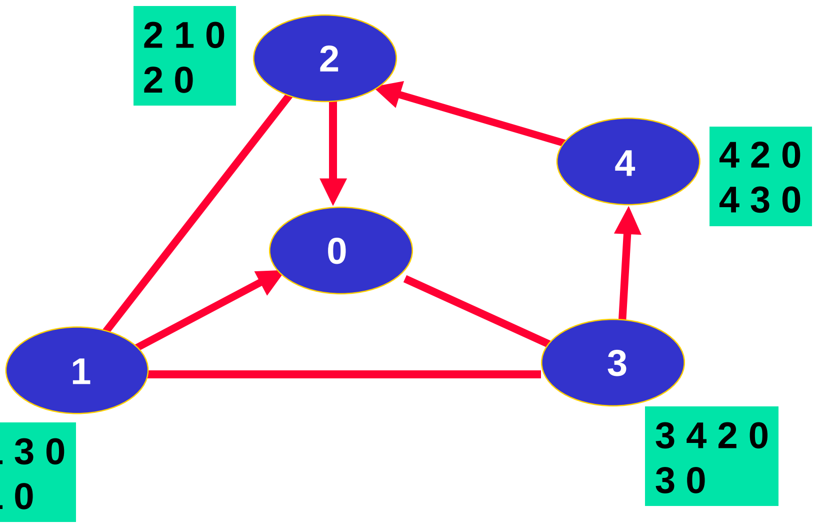
Example 1



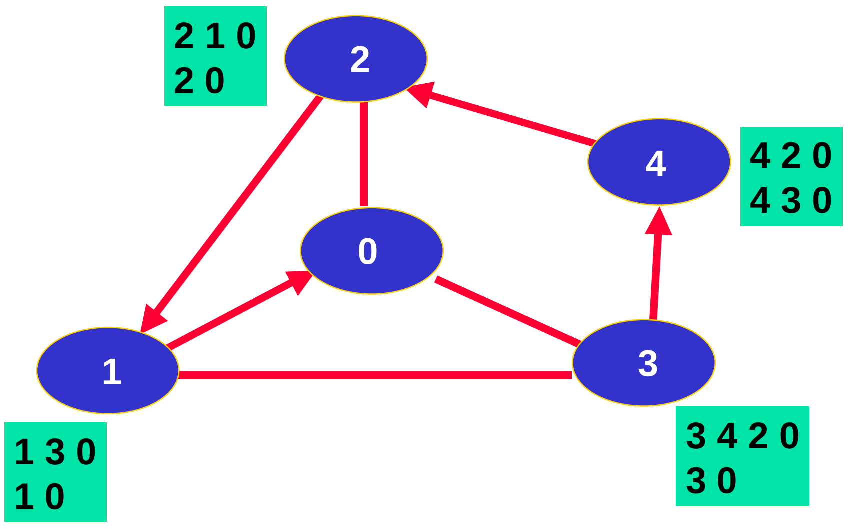
Example 2



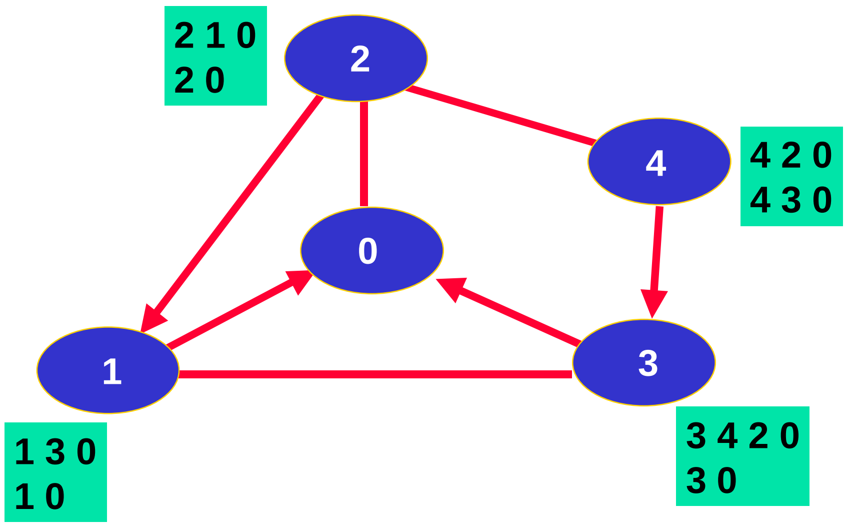
Example 2 (contd.)

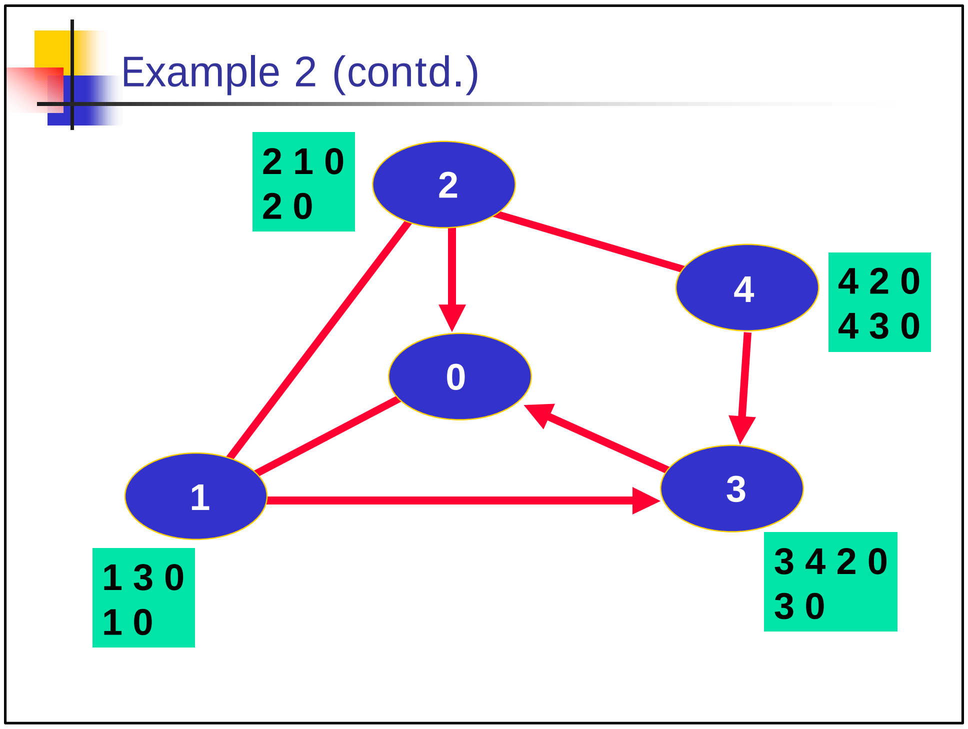


Example 2 (contd.)

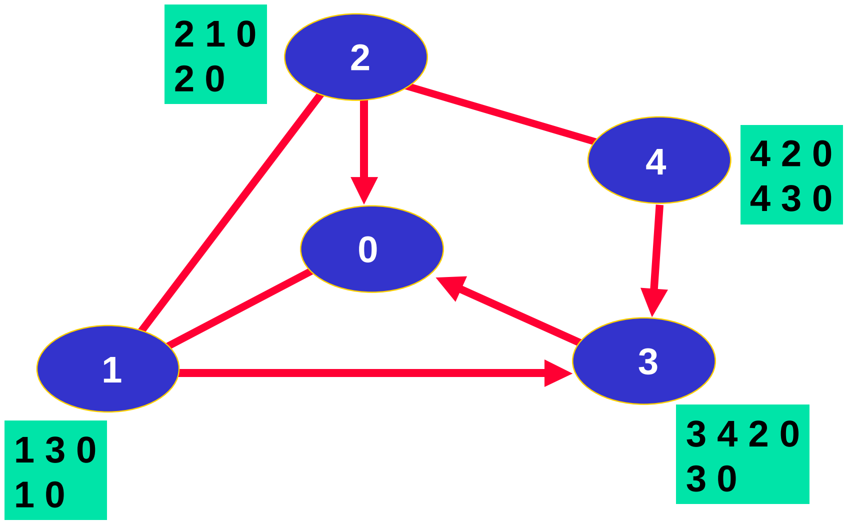


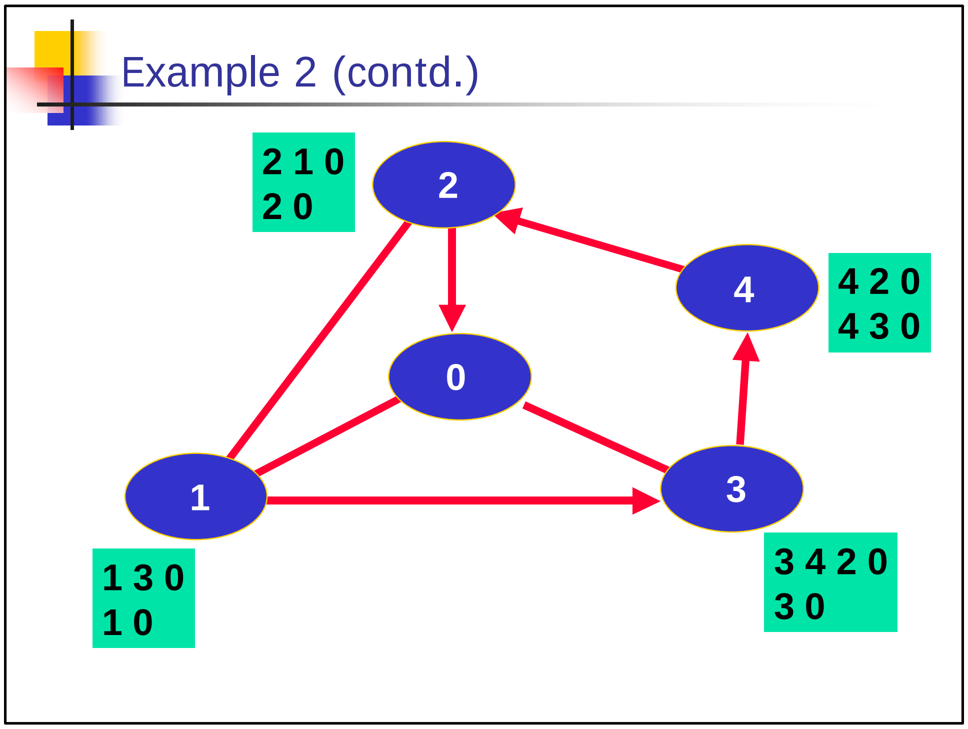
Example 2 (contd.)



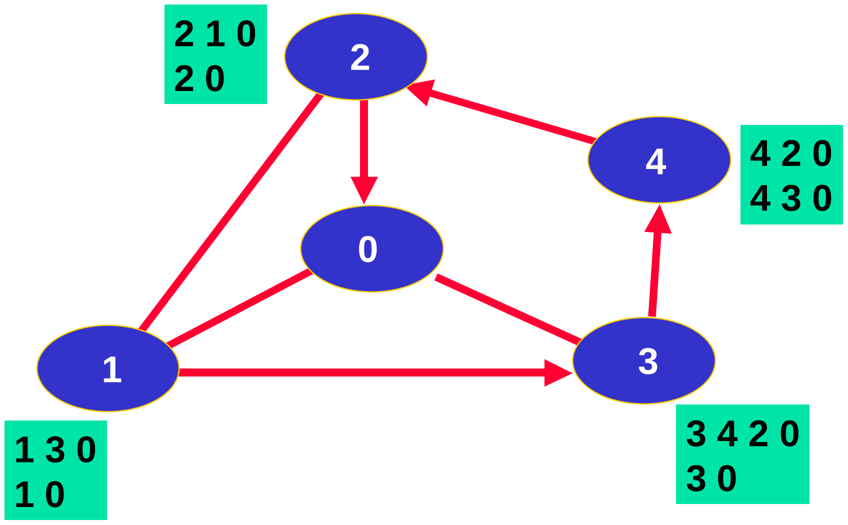


Example 2 (contd.)

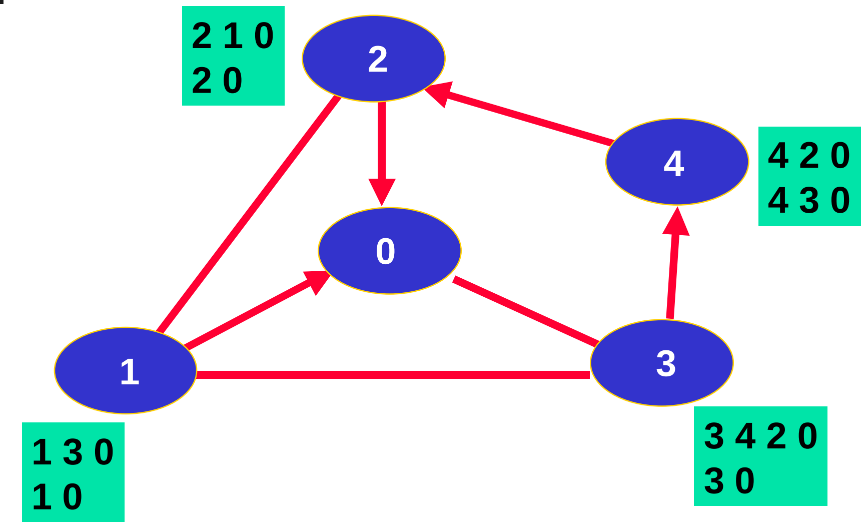




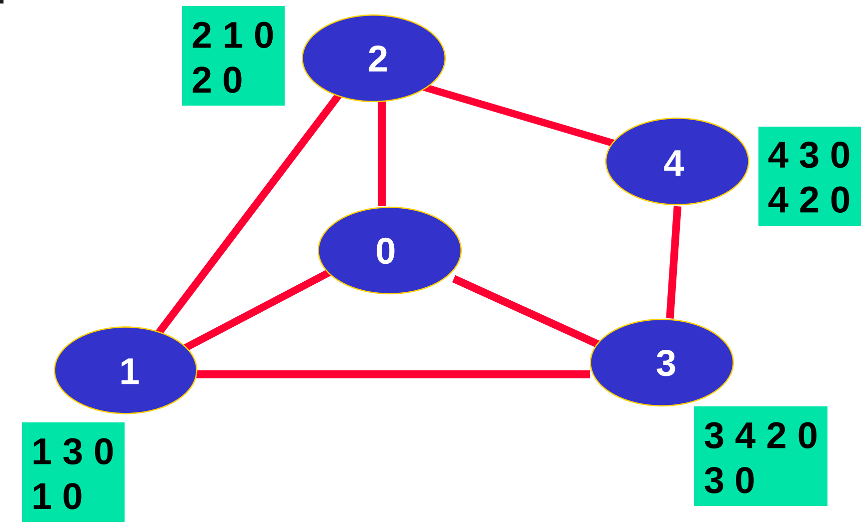
Example 2 (contd.)



Example 2 (contd.)



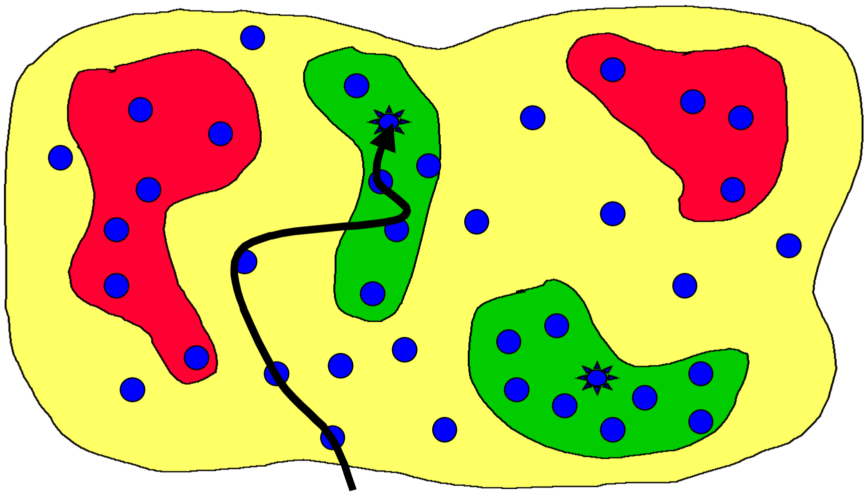
Example 3



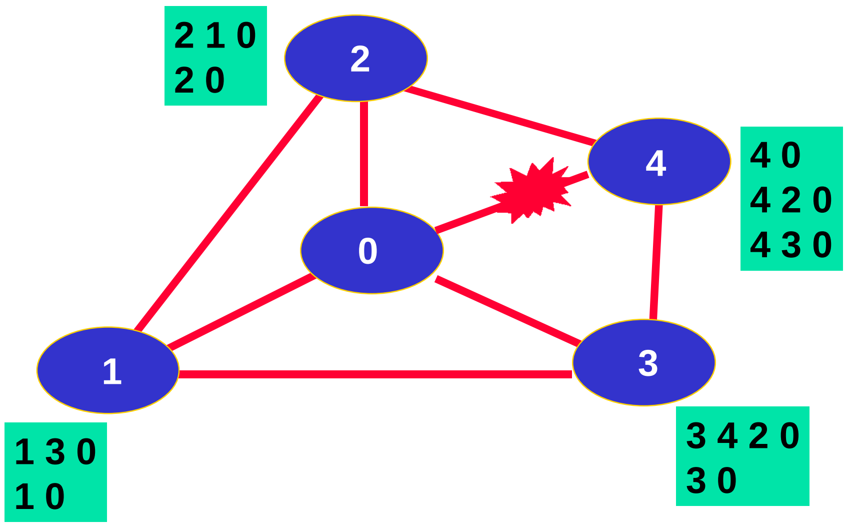
SPVP wanders around assignment space

● = assignment

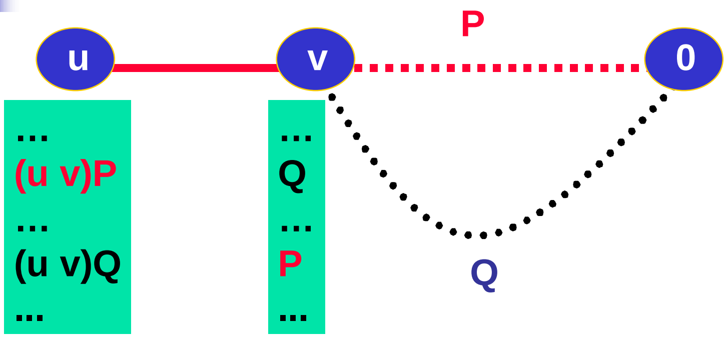
✸ = solution



Unsolvable after link failure



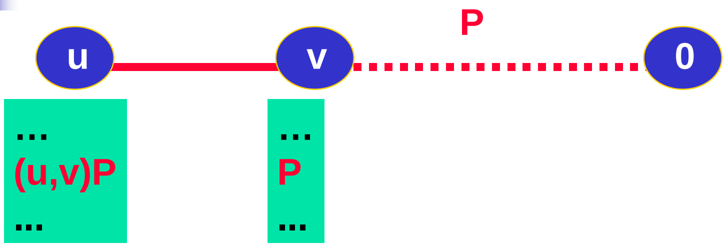
Dispute Digraph



Gives the *dispute arc*

$$\boxed{Q \longrightarrow (u\ v)P}$$

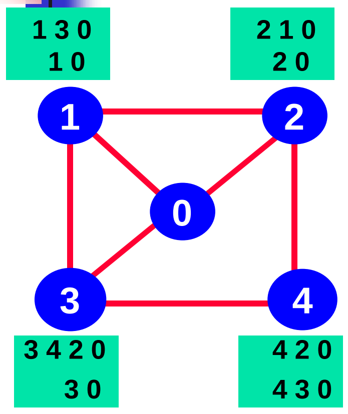
Dispute Digraph (cont.)



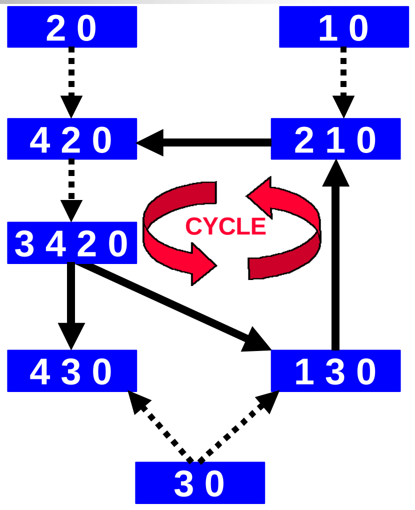
Gives the *transmission arc*

$$\boxed{P \dashrightarrow (u,v)P}$$

Dispute Digraph example



BAD GADGET II



Sufficient Condition for Sanity

If an instance of SPP has an <u>acyclic</u> dispute digraph, then

Static (SPP)	Dynamic (SPVP)
solvable	safe (can’t diverge)
unique solution	predictable restoration
all sub-problems uniquely solvable	robust with respect to link/node failures