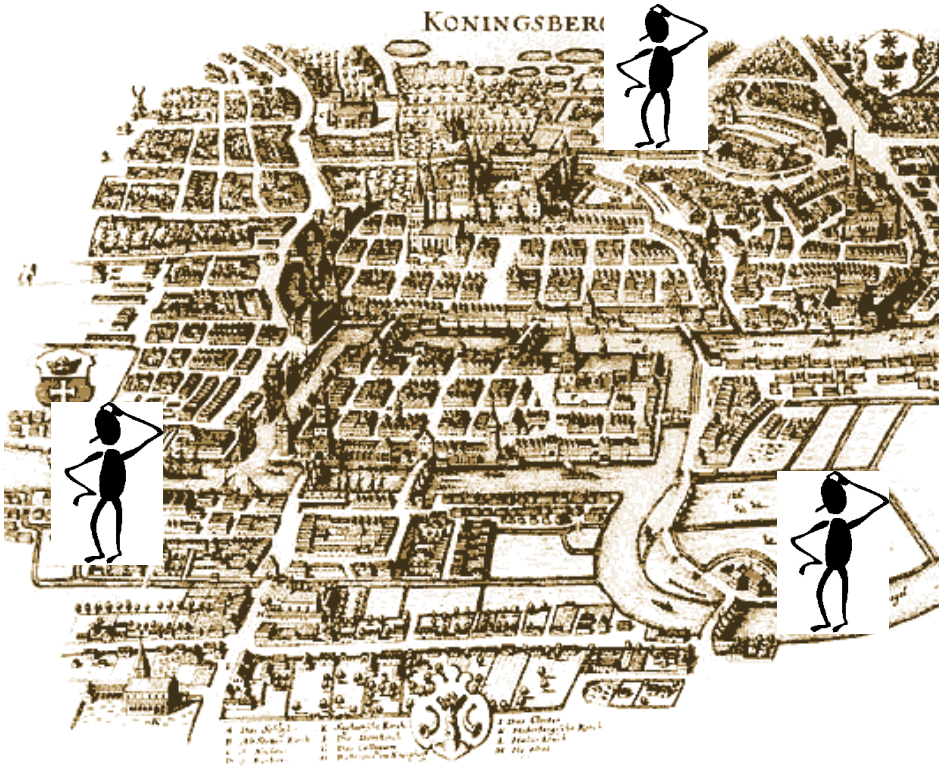
Symmetric Rendezvous in Graphs: Deterministic Approaches

Shantanu Das
Technion, Haifa, Israel
http://www.bitvalve.org/~sdas/pres/Rendezvous_Lorentz.pdf

Coauthors:
Jérémie Chalopin, Adrian Kosowski, Peter Widmayer

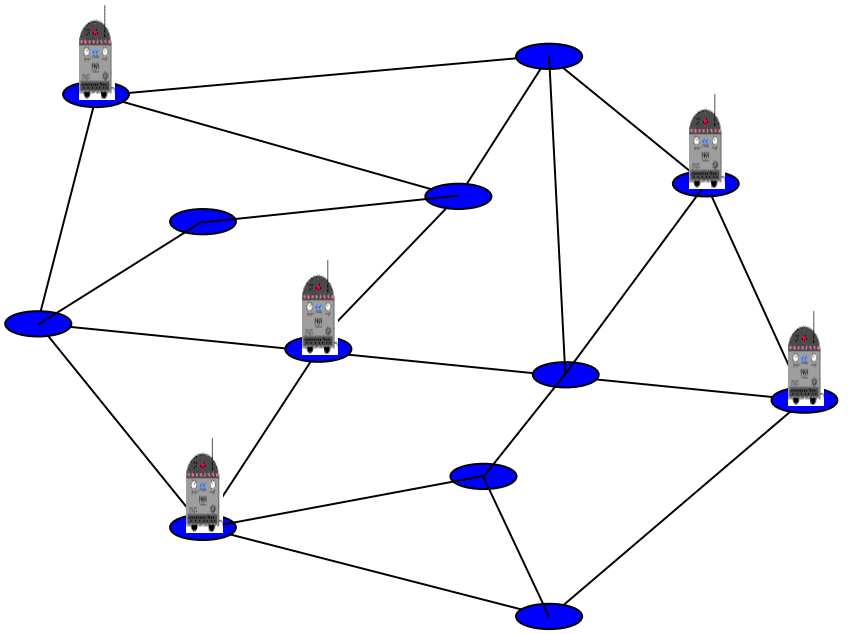
What this talk is about...

A group of friends visiting an unknown town gets separated. How can they get together again?



What this talk is about...

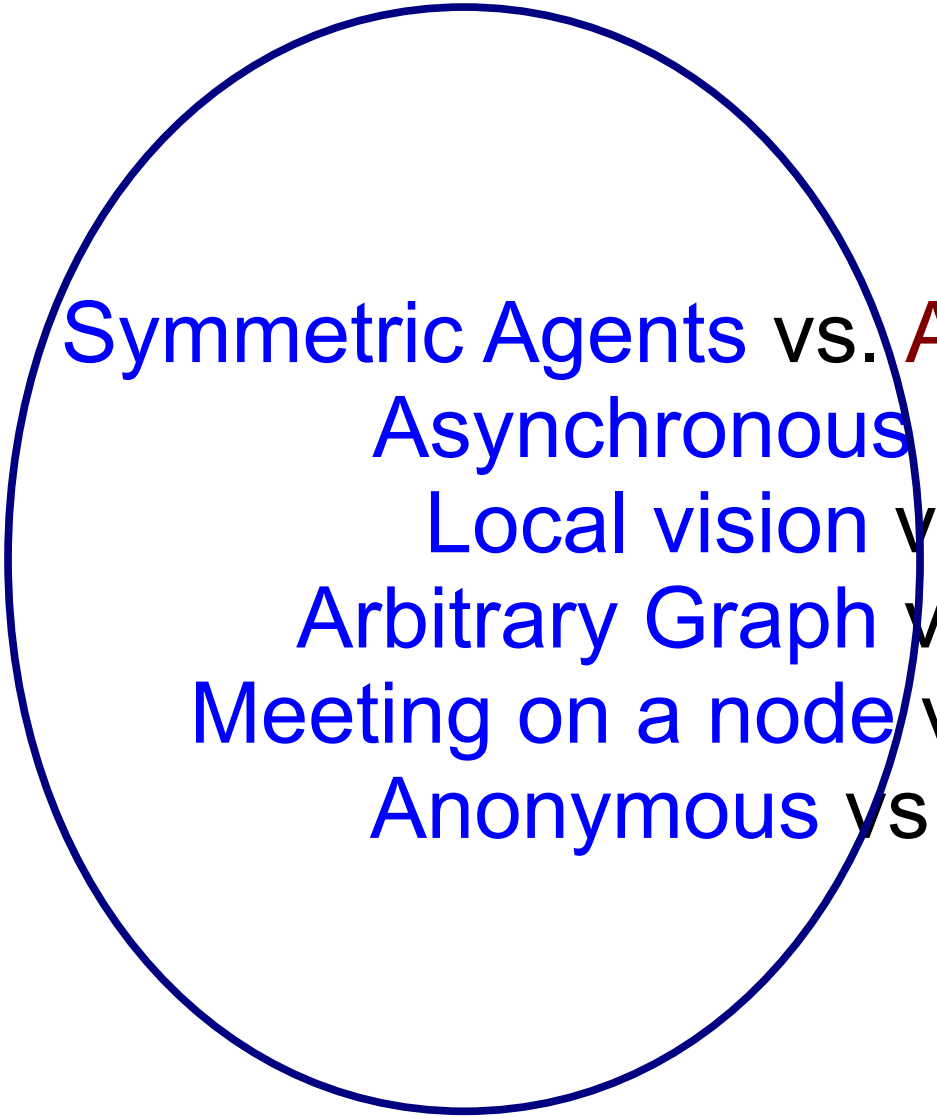
- *Identical* Mobile Agents in finite graphs
- Deterministic Algorithms



Objective: Must meet at a node!

Optimize: Worst case complexity

Mobile Agent Models



Symmetric Agents vs. Asymmetric (Distinct ID's)
Asynchronous vs. Synchronous
Local vision vs. Global vision
Arbitrary Graph vs. Known topology
Meeting on a node vs. Meeting anywhere
Anonymous vs. Labelled Nodes

Outline of the Talk

- Mobile agents and their capabilities
- When is Rendezvous possible?

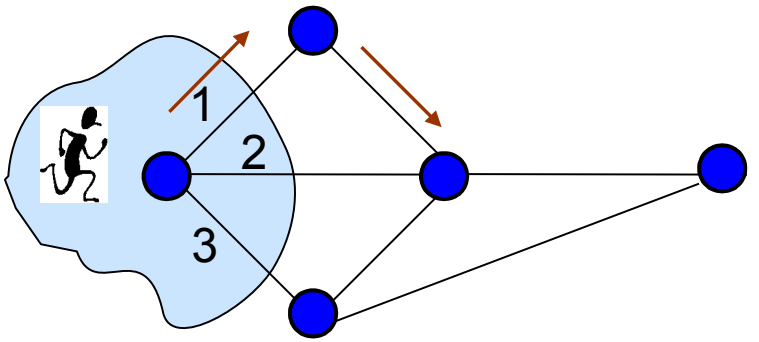
- Rendezvous Algorithms, Tradeoffs
  - Without any Marking
  - Marking on Nodes
  - With Tokens only

- Fault Tolerance
  - Faulty Tokens
  - Faulty nodes/edges

- Future Directions

Mobile Agent Model

- Operates in Asynchronous LCM cycles
  - Look-
  - Compute-
  - Move (1 edge at a time)



- Local Orientation
  (Incident edges are ordered)
- Local Visibility (#agents, tokens)

- Memory (non-constant)
- Prior Knowledge (n,k,D, N>n etc.)

- Marking of nodes
  YES / NO
- Deterministic symmetric strategy
  (no coin-flips, no ID's)

The environment

An instance of the problem is given by:

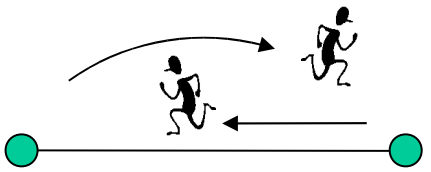
$$(G, \lambda, \delta, \xi, \chi_p)$$

- Finite connected Graph $G(V,E)$
- Nodes have labels $\lambda: V(G) \rightarrow L$ (may not be bijective)
- Port numbering: δ
- Set of Agents: $\boldsymbol{\xi}$
- Locations of Agents $p: \xi \rightarrow V(G)$
  $\Rightarrow \chi_p: V(G) \rightarrow \{0,1\}$

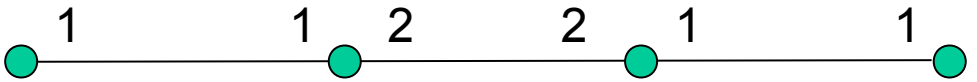
$|\boldsymbol{\xi}| = \mathbf{k}$, $|V(G)| = \mathbf{n}$, $|E(G)| = \mathbf{m}$, Max.degree(G)= $\mathbf{d}$

Impossibility of Rendezvous

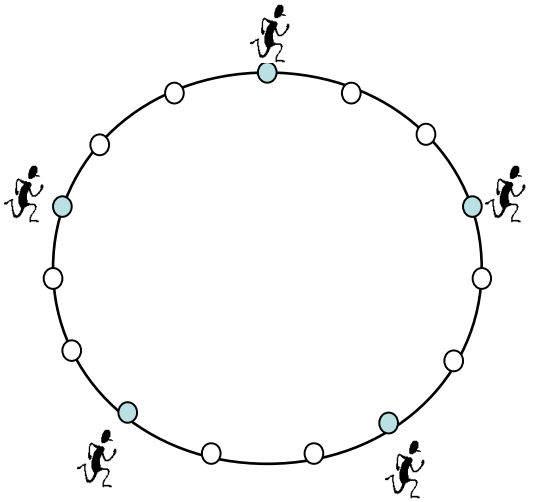
- Single Edge



- Symmetric Line



- Symmetric Ring



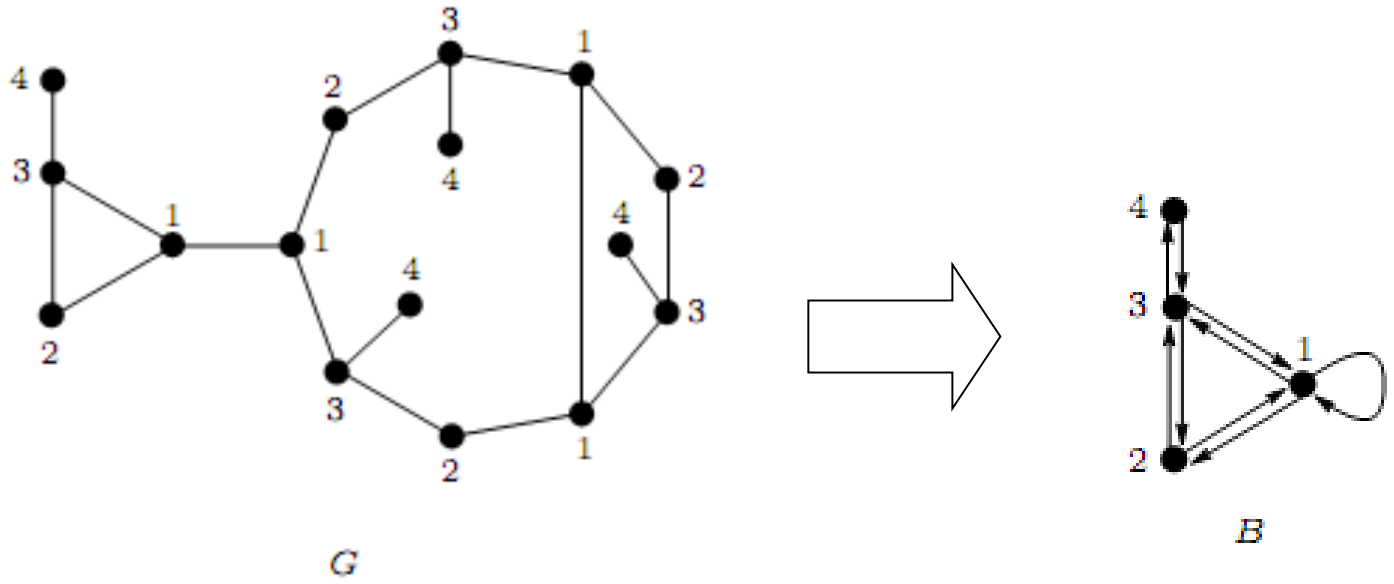
Rendezvous-with-Detect

- Explicit Termination
  - Solves Rendezvous whenever deterministically possible
  - Otherwise output NO

- Cost Measures
  - *Move Complexity* (1 edge traversal costs 1)
  - Agent Memory
  - Node Memory
  - Number of Tokens

When is Rendezvous feasible?

Theorem: [Boldi & Vigna 2001]
Rendezvous is possible only if **(G, λ, δ, χ_p)** is covering-minimal.



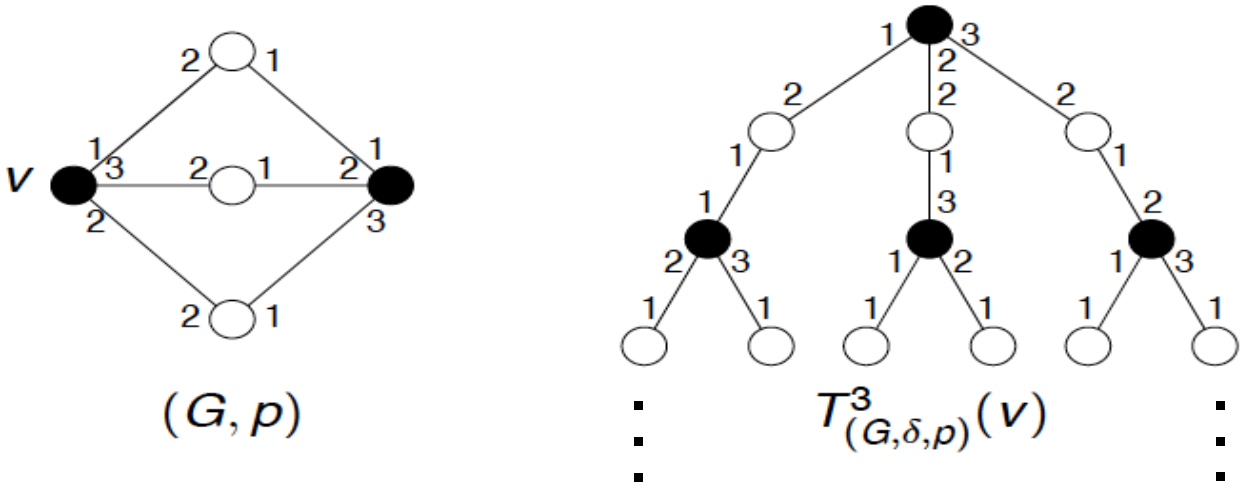
The homomorphism is locally bijective & preserves labels ***λ, δ, χ_p***

Minimum-base: The smallest graph B such that G *covers* B

When is Rendezvous feasible?

Definition: [Yamashita & Kameda 1988]

- The view of a node v in (**G, δ, p**) is the infinite tree of all paths starting at v.



Nodes having identical views are symmetric.

[Norris 1995] View of depth **n-1** is sufficient !

Algorithms

for

Rendezvous

Rendezvous without Marking

- Number of agents and their initial locations is irrelevant!
- Rendezvous is feasible only if (**G, λ, δ**) is covering minimal.

Algorithm 1:

- Construct the minimum-base graph B
- If |B| = n, meet at vertex 1 of B.
- Otherwise output “Not solvable”;

Algorithm-1 solves Rendezvous-with-Detect.

How to map an (anonymous) graph without marking?

Mapping an unlabeled Graph

Theorem: [Aleliunas et al. 1979]
For any positive integers n, d, d < n, there exists a *universal exploration sequence* for the family of all graphs with at most **n** nodes and maximum degree at most **d**.
A UXS is sequence of edge labels that when applied to any graph G (in the family) is guaranteed to visit every node of G

- Mapping Algorithm
  For i=1 to n
  - Traverse G using uxs(n,d);
  - Classify the visited vertices according to View[i]
  - Compute distinguishing paths of length i

- Cost = O(n^6 d^3 log n) moves = O(n^3d) * |uxs(n,d)|

Rendezvous without Marking

To summarize:

- Algorithm 1 solves *Rendezvous-with-Detect*, when n is known.

- If the agents only know a bound **N ≥ n**, then the algorithm solves *Rendezvous* for the solvable cases.

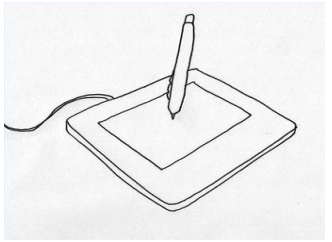
- Can we improve the cost of the algorithm? O(n^3 d^6 log n)

Rendezvous With Marking

Rendezvous with Marking

[Baston & Gal 2001] RV with marks on Starting locations!

Rendezvous is feasible only if and only if
($\mathbf{G}, \boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\chi}_p$) is covering minimal.

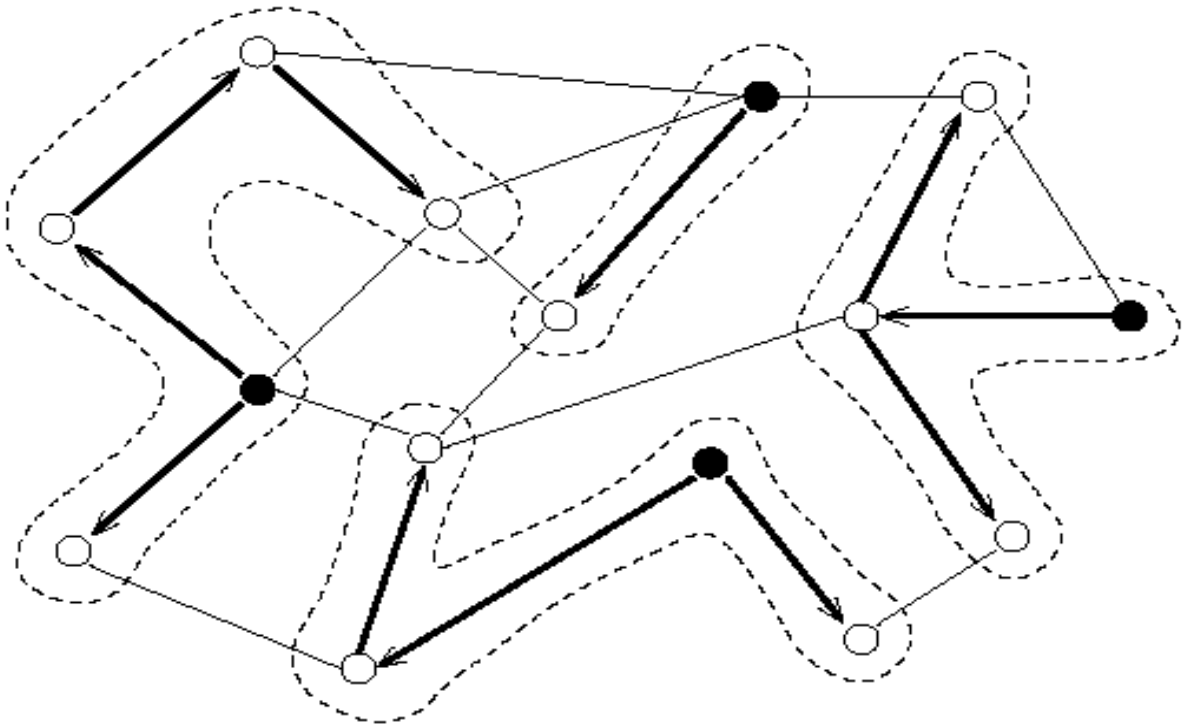


Whiteboard Model:
- Each node v has a whiteboard.
- An agent on node v can write any information on it.

Rendezvous with Marking

Distributed DFS:

- The agent that first reaches node v, claims it by marking on it.
- Spanning forest of the graph (with k components)
- How to merge the components?

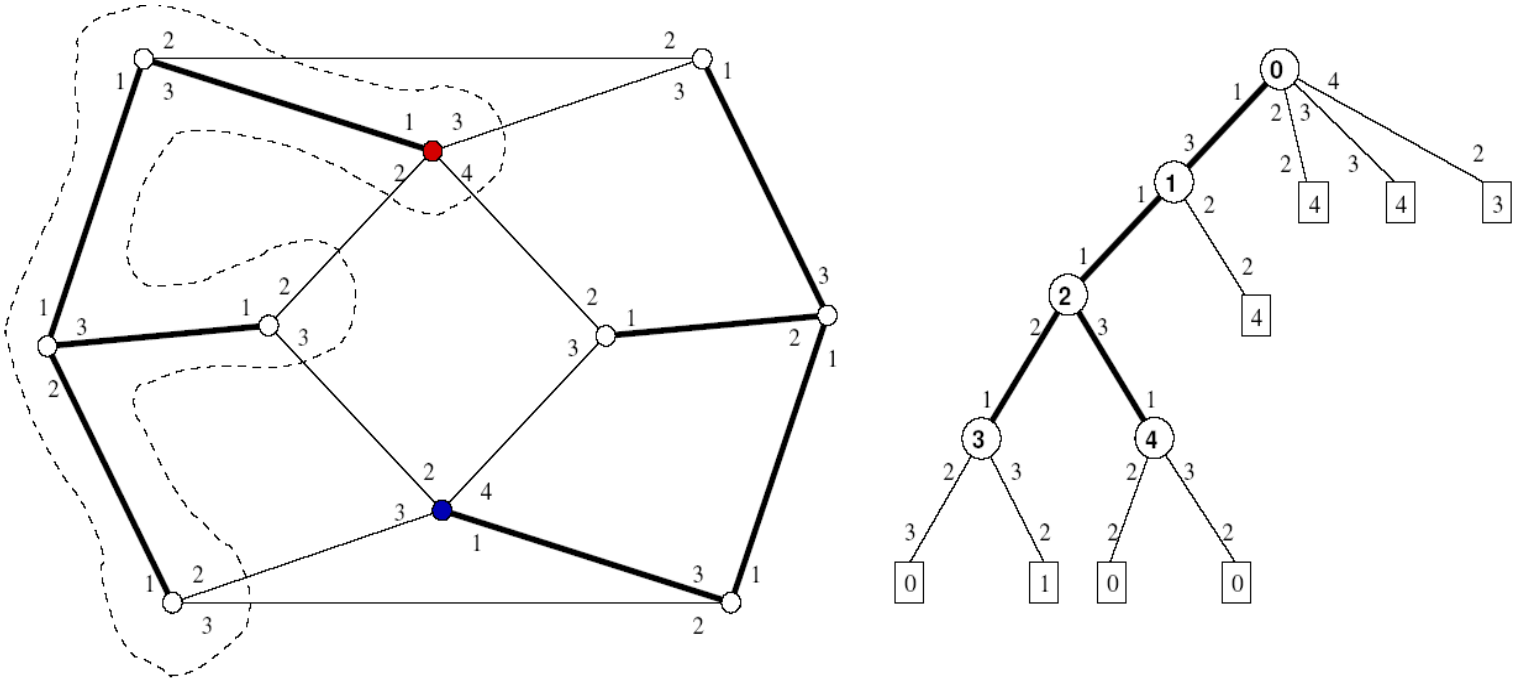


Partial-View of an agent

The *Partial-View* of an agent is the territory T of the agent plus some external nodes corresponding to each outgoing edge.

If the Partial-views of all the agents are identical then $(G, \lambda, \delta, \chi_p)$ is not covering minimal.

=> Rendezvous is not solvable!



Rendezvous with Marking

Algorithm-2:

- Perform DDFS and mark your territory;
  (Let T_A be the territory marked by agent A)
- Repeat (for at most k phases)-
  - Compare your Partial-View with neighbors;
  - If lost, become passive and stop;
  - If won, acquire the territory of loser and continue;

Until (T_A spans the whole graph G);

Correctness of the Algorithm:

In each phase,

- either at least one agent loses, or, all agents have identical PV's.

After k phases,

- either there is single Tree, or, Rendezvous is impossible;

Cost = **O(m k)** moves

Node Memory = **O(m log n)** bits

Optimizations and Tradeoffs

Problem	Move-Cost	Node Memory	Agent Memory
RV-with-Detect	O(m k)	O(m log n)	O(m log n)
RV-with-Detect	O(m^2 k)	O(log n)	O(m log n)
RV-with-Detect	O(m^2 n k)	O(m log n)	O(log n)
Rendezvous	O(m log k)	"	"

Rendezvous Using only Tokens

Rendezvous using Tokens

Algorithm-3:
- Mark your home with a Token;
- Construct the minimum-base graph B.
- If |B| = n, meet at vertex 1 of B.
- Otherwise output “Not solvable”;

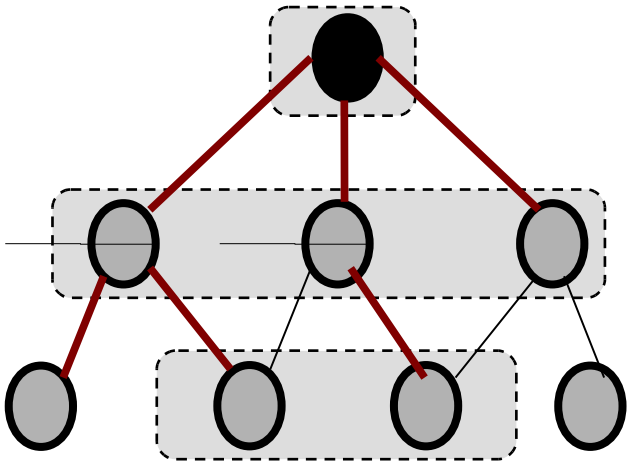
} Algorithm-1

Algorithm-3 solves *Rendezvous-with-Detect*.
Cost = O(n^6 d^3 log n) moves
Number of tokens = 1 per agent.

How to obtain a more efficient algorithm?

More efficient Rendezvous

Algorithm BFST:
- Mark your home with a Token;
- Perform BFS traversal and store paths in *RootPaths*;
- For each visited vertex v, and each p in *RootPaths*,
  - Check if Target(v,p) is a marked homebase;
    - If no path leads to home, add v to the BFS tree;
    - Else add a cross edge to the map;
- Stop when there are no more edges to explore;



For k=2 agents,
If maps of the agents are identical then
$(G, \lambda, \delta, \chi_p)$ is not covering minimal => Rendezvous is not solvable!
Otherwise, compare the maps and choose one of the root nodes.
Thus we can solve Rendezvous-with-Detect.

Cost = O(n^2 d^2) moves
Number of tokens = 1 per agent.

Rendezvous with Marking

- Move Cost:

Rendezvous-with-Detect can be solved in O(m k) moves.

- Agent Memory:

Agent memory of O(log n) is sufficient to solve Rendezvous-with-Detect.

- Number of Tokens:

One token is sufficient to solve Rendezvous-with-Detect.

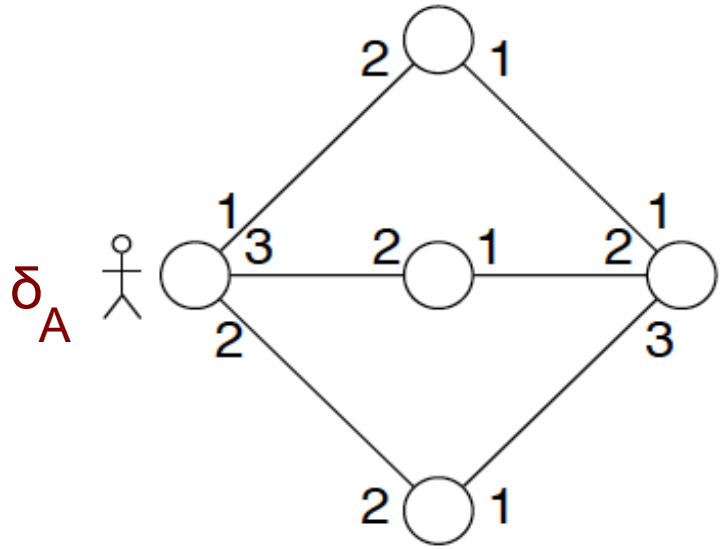
When does having more tokens help?

SPECIAL CASE:

Rendezvous Without Common Port-Numbering

RV without common Port-numbers

What if the agents do not agree on local orientation?



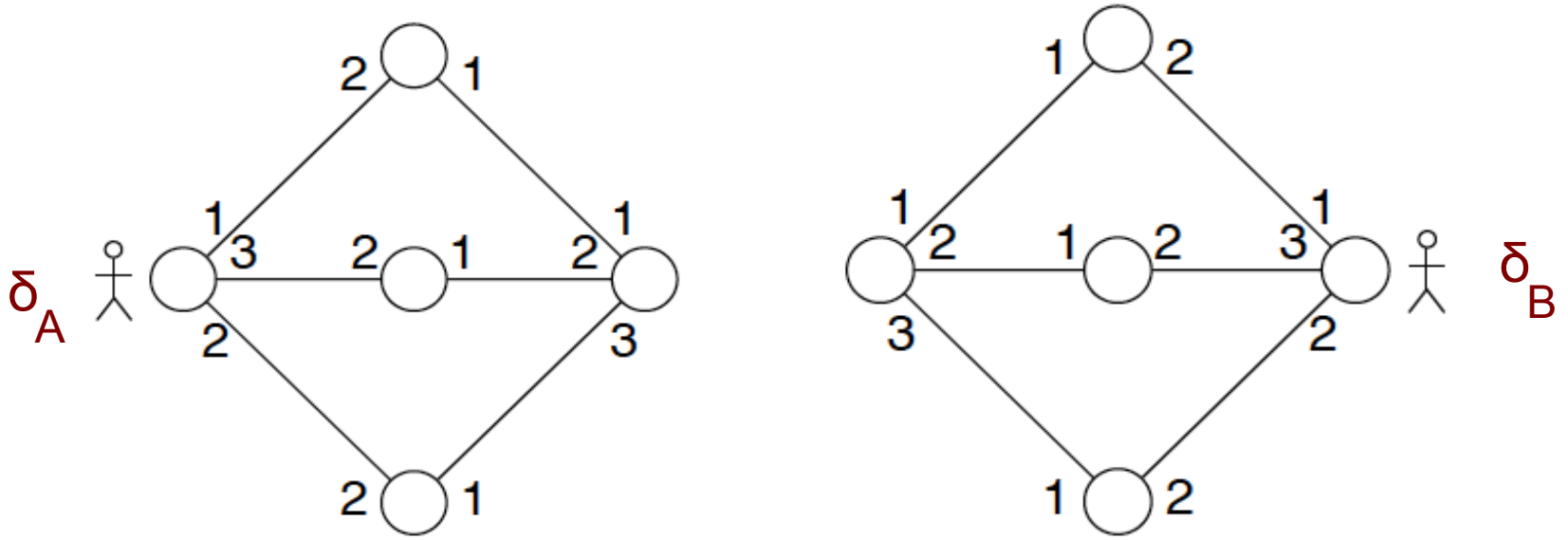
An agent must be able to order the edges incident to any node (in a consistent manner).

Otherwise the agent cannot navigate in G!

RV without common Port-numbers

What if the agents do not agree on local orientation?

- It is not possible to uniquely order the views
- Which instances are solvable in this case?
- Different techniques are needed for solving RV in this case.



RV without common Port-numbers

Theorem: If Rendezvous can be solved in (G,λ,δ,χp) for any common port-numbering δ (chosen by adversary), then Rendezvous can be solved in (G,λ,χp) without agreement on port-numbering.

(*Assumption:* Each agent has 2 tokens!)

Assuming that (G,λ,χp) is covering minimal...

- *Each agents can obtain a map of the graph.*
- *Each agent knows the homebases of the other agents (marked by token).*
- *The maps of the agents are isomorphic (no unique ordering of the nodes).*

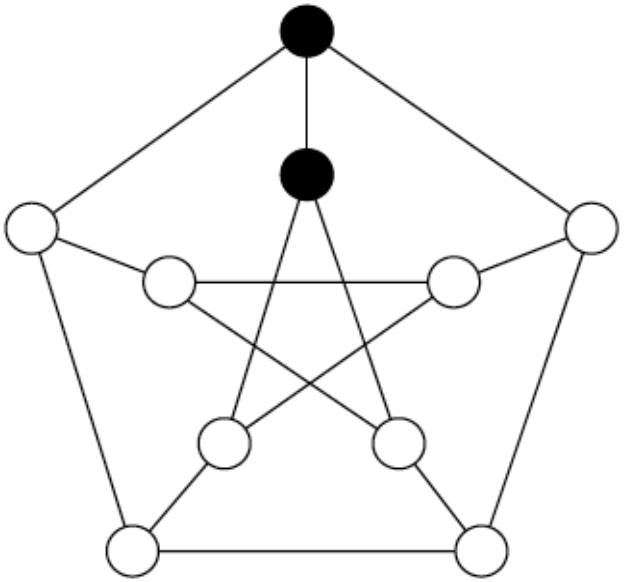
Equivalence classes of V(G)

Definition: Equivalent vertices

$v \sim w$ if there exists an automorphism σ of (G, p) such that

$$\sigma(v) = w$$

We only consider automorphisms that preserve homebases.



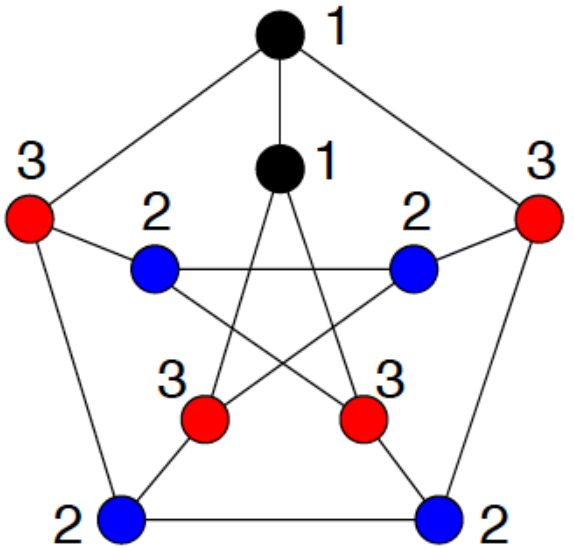
Equivalence classes of V(G)

Definition: Equivalent vertices

$v \sim w$ if there exists an automorphism σ of (G, p) such that

$$\sigma(v) = w$$

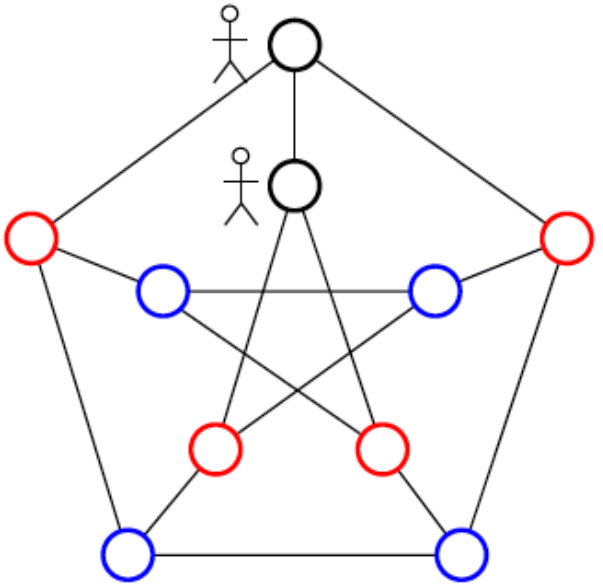
We only consider homomorphisms that preserve homebases.



Refining the Class Partition

Basic Technique:

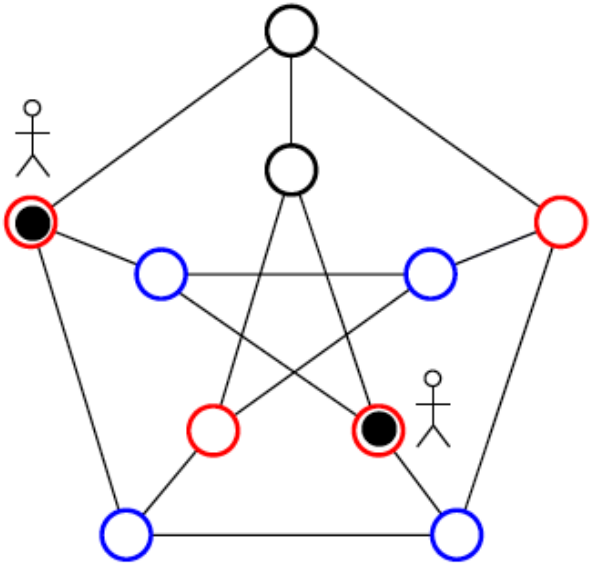
- Mark nodes to break symmetry within a class and
- Refine the class partitions.



Refining the Class Partition

Basic Technique:

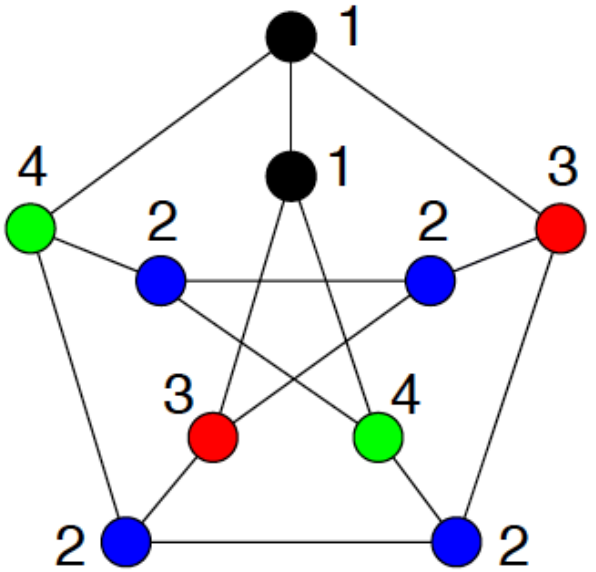
- Mark nodes to break symmetry within a class and
- Refine the class partitions.



Refining the Class Partition

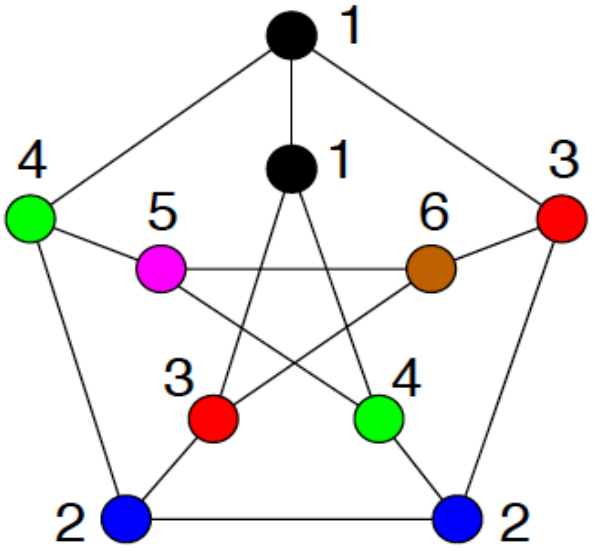
Basic Technique:

- Mark nodes to break symmetry within a class and
- Refine the class partitions.



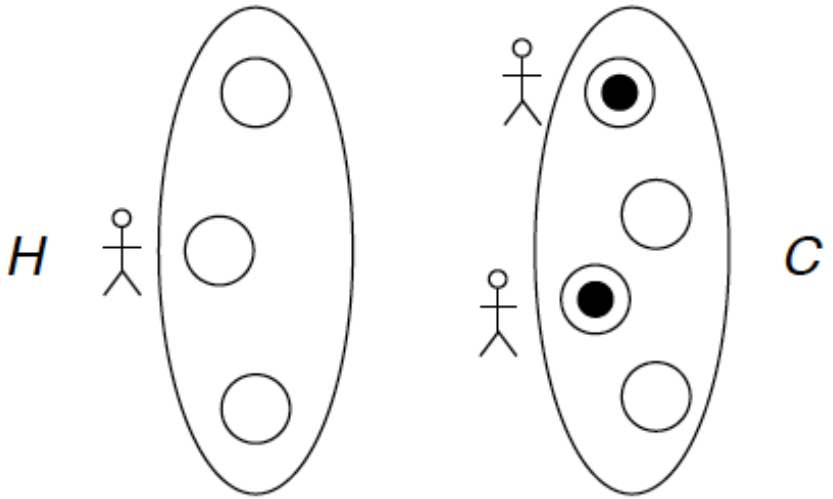
Refining the Class Partition

- As long as there are classes of different sizes, it is possible to break a larger class into two.
- The process ends when *all classes have the same size*.



Synchronizing the Agents

- It is not easy to mark vertices in an asynchronous setting.
- We divide the algorithm into rounds.
- In each round one class C of nodes is chosen to be partitioned.
- The start and end of each round is signaled adding or removing tokens to the homebases.



Proof of Correctness-1

- **Claim 1:** The classes $C_1, C_2, \ldots C_t$ obtained by the algorithm are all of size 1.

Assume that the classes have multiple nodes (i.e. t < n) ...

Theorem [Yamashita and Kameda '99]

Rendezvous can be solved on (G, p) *if and only if there is no partition* $C_1, C_2, \ldots C_t$ *with* $1 \leq t < n$ *such that*

- $G[Ci, Cj]$ *is regular*
- $G[Ci]$ *is d-regular for some d, and contains a perfect* matching if d is odd

- The first property can be shown using a counting argument.
- The second property follows from a lemma of Gallai-Edmonds about vertex-transitive graphs.

Proof of Correctness

- Claim 1: The classes $C_1, C_2, \ldots C_t$ obtained by the algorithm are all of size 1.
- Claim 2: The agents compute the same ordered sequence of classes $C_1, C_2, \ldots C_t$.

To Summarize:

- *The algorithm solves Rendezvous without agreement on local orientation for all instances (G,p) for which Rendezvous is solvable with common local orientation*.
- We also show that two tokens per agent are necessary and sufficient for the former case while one token suffices in the latter case.

Fault Tolerant

Rendezvous

When Tokens Fail ...

Token Failures:

- The token placed on a node may disappear suddenly.
- Failures are permanent.

(tokens eaten by birds, lizards, etc.)

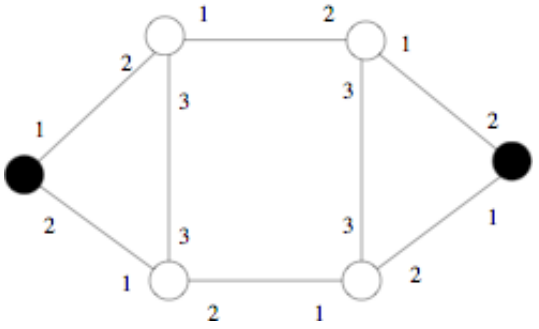
[Flocchini et al. 2004]: Ring Networks

(for special values of **n** and **k)**

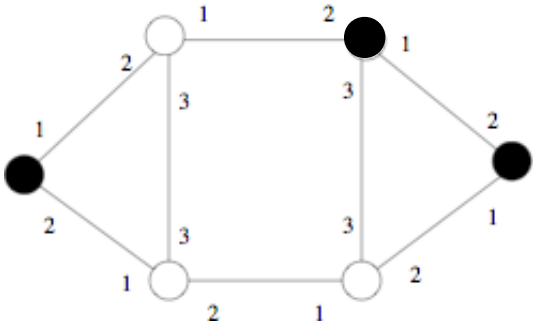
Theorem: Rendezvous can be solved in (G,λ,δ,χp) with one token per agent, in presence of **`f < k`** faults, iff it can be solved in the fault-free case.

(*Assumption:* Both **n** and **k** are known!)

RV using Tokens



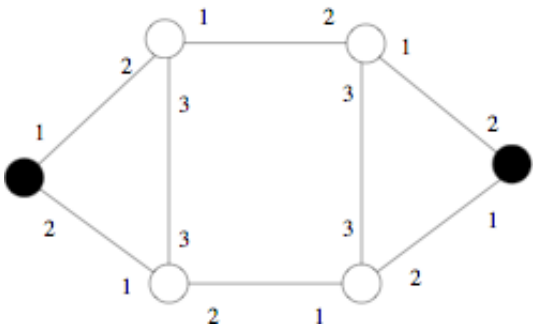
RV is impossible



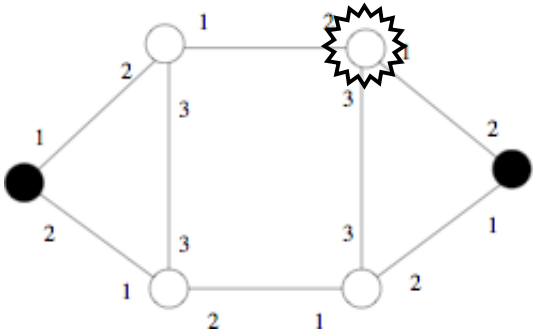
RV is Solvable

RV using Tokens

When tokens fail:



RV is impossible



RV is Solvable ??

RV in spite of Token Failure

Basic Idea:

- Each agent puts the token in its home.
- Each agent builds a map of G and returns to its home.
- If the token at home is still present, the agent remains at home, "guarding" the token. (*Owner*)
- If the token at home has disappeared, the agent becomes a *Runner*.

- *If a token guarded by an agent fails, it has no effect on the execution of the algorithm.*

- *If the agent fails to build a map due to some token failures, the agent can detect this fact.*

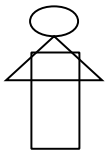
n'/k' ≠ **n/k** => *Token Failure!*

RV in spite of Token Failure (cont.)

3 types of agents:

RUNNER agents: Those agents which have lost their token; They traverse the ring collecting information.

OWNER agents: Those agents who have not lost their Token in the first round. They remain stationary, guarding the token.



LEADER agent: Exactly one owner becomes Leader, (only if there are no Runners).

When **f** > **1** Tokens fail

There is at least 1 RUNNER and 1 OWNER!

Each RUNNER associates with the nearest OWNER.

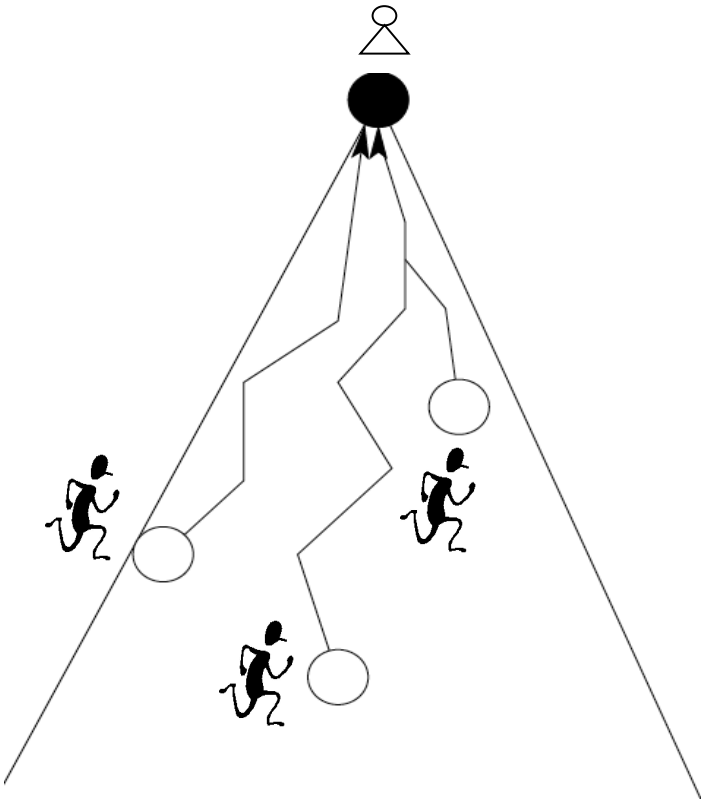
(it remembers the path to home)

<u>OWNER</u>:

- remains stationary and,
- receives RUNNERs.

<u>RUNNER</u>:

- Traverses the graph G
- communicates with OWNERs.



An OWNER agent obtains complete information to construct a map of (G,λ,δ,χp)

When **f=0** (no token fails)

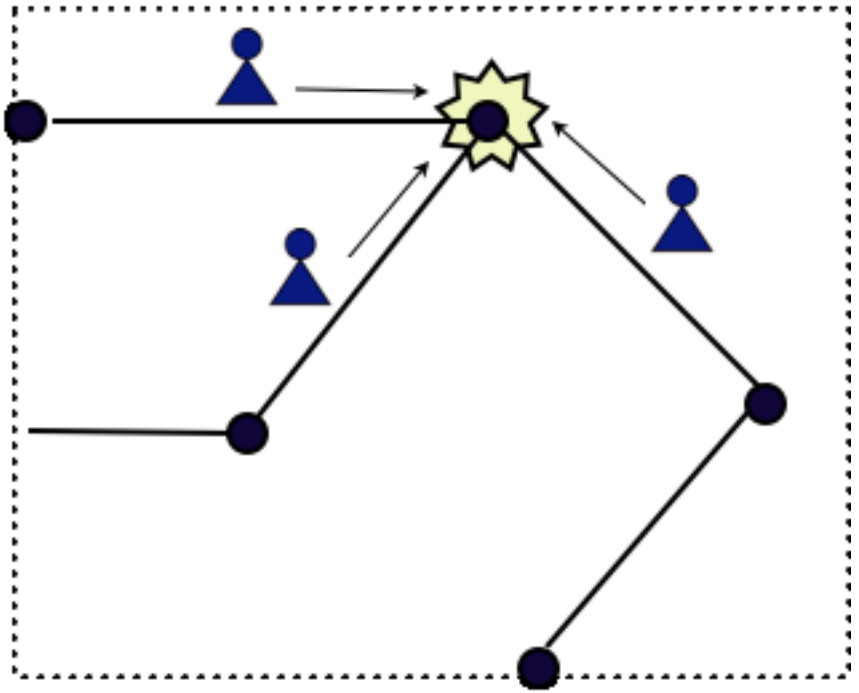
- Each agent becomes an OWNER after first round.
- Each agent builds a map and returns to its home.
- If RV is not solvable, every agent detects it!
- Otherwise,
  - The agent at the RV-location becomes LEADER.
  - The LEADER checks if there are **(k-1)** OWNER agents.
  - The LEADER collects the other agents to RV-location.

The algorithm solves Rendezvous-with-Detect in the presence of $0 \leq f < k$ faults occurring at any instance during the execution.

Cost of the algorithm: (same as Algorithm-1).

Other Faults : Black-Holes

- Faulty Node: destroys any agent arriving at that node. (*Black Hole*)
- Faulty Edge: an agent attempting to traverse the edge is destroyed.
- Faults are permanent and do not occur during the execution.

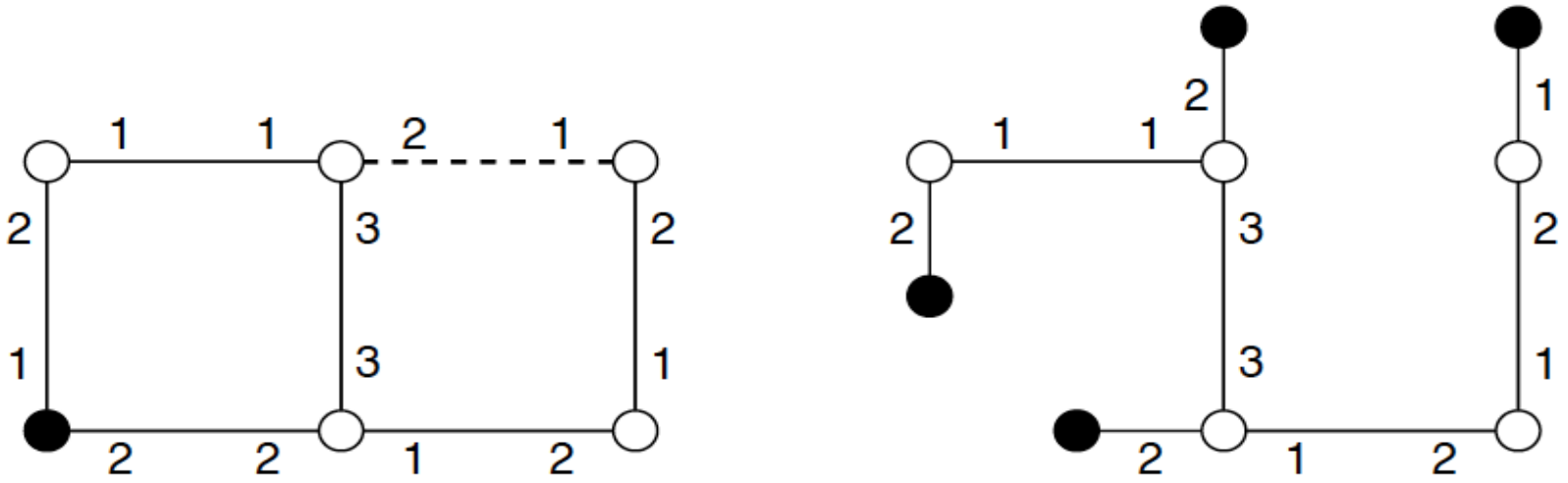


[Dobrev et al. 2001]: Ring Networks
(With one Black Hole)

Rendezvous in spite of Black hole faults:

- Agent start at safe nodes.
- The subgraph consisting of safe nodes and edges is connected.
- Partial Rendezvous: Gather as many agents as possible!

Extended-Map of a Faulty Graph

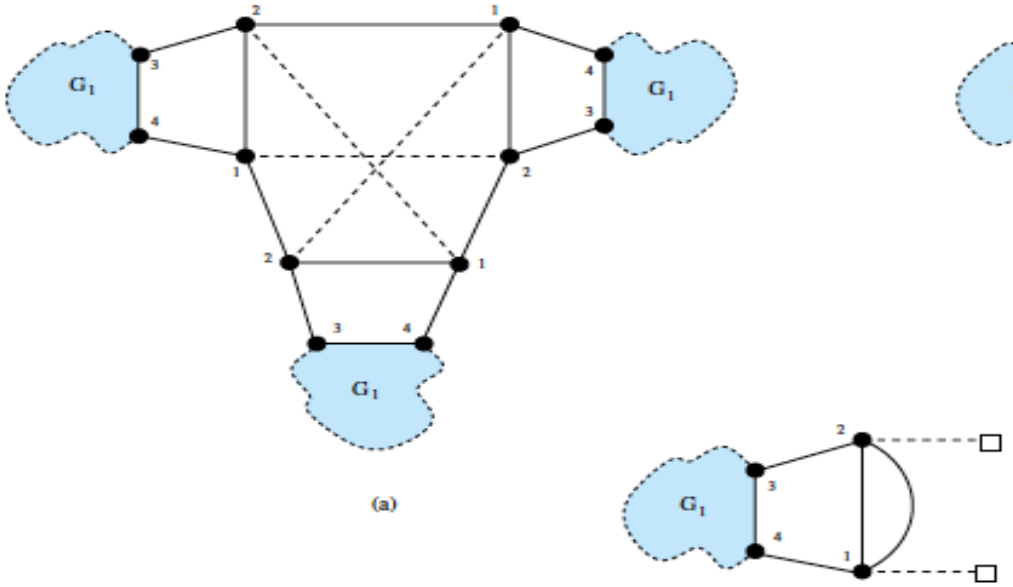


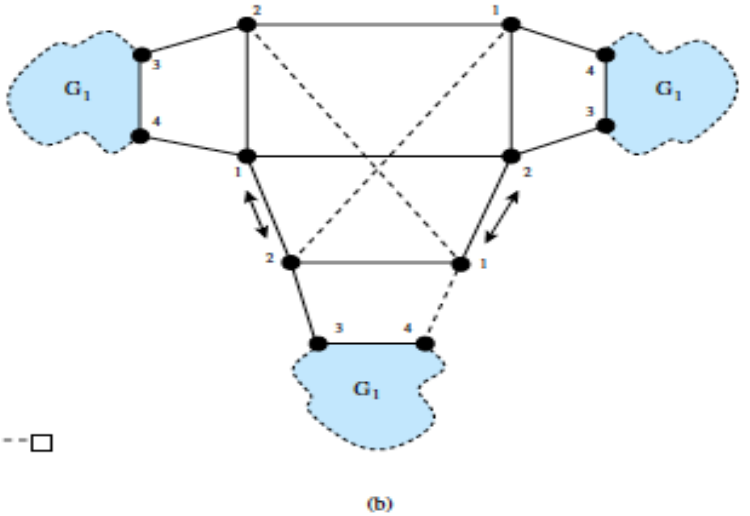
- Number of Faulty Links: t = 4

Theorem: We can rendezvous at most (k-t) agents.

Rendezvous in Faulty Graphs

- **Theorem**: Rendezvous of k-t agents can be solved <=> the extended-map is covering minimal!
- The agents need to know **n** or a tight bound B < 2n
  (k and t may be unknown)
- Rendezvous-with-Detect is not possible, even if the topology is known





Exploring Graphs with Faults

- How to traverse a faulty graph?
- *Cautious Walk* Approach:
  - To visit an unexplored edge (u,v) mark it as “Dangerous”
  - Traverse e and return back to u
  - Mark edge (u,v) as “explored” (safe!)
- Ensures that no more than t agents die!

Note: Agents are asynchronous! It is not possible to distinguish a faulty edge from a slow edge.

Rendezvous in Faulty Graphs

The Algorithm RVBH:

- Alternating phases of exploration and competition
- Each team of agents has a territory that they try to expand
- *Exploring phases*
  - Agents explore edges that have never been traversed yet
  - Some agents may die during Exploration
- Competing phases
  - Team of agents that have neighboring territories try to invade each other
  - No agent can die during Competition
  - For each team, one competing agent is enough
- The algorithm terminates when one team acquires more than n/2 nodes.

Rendezvous in Faulty Graphs

- Algorithm RVBH solves *Maximal-Rendezvous* whenever the extended-map is covering minimal.
- Cost of the algorithm = O(m(m+k)) moves.
  - There are O(m+k) competing rounds.
  - Total cost of competing rounds = O(m(m+k))
  - Total cost of exploring rounds = O(n(m+k))
- The algorithm has optimal move complexity.

Theorem: Any deterministic algorithm for maximal rendezvous in faulty graphs with at most m edges, requires O(m(m+k)) moves in the worst case.

Conclusions

- We can solve Rendezvous-with-Detect under minimum assumptions about the environment and even overcoming failures.
- The algorithms work for connected graphs of arbitrary topology.
- What happens when the environment is a directed graph? (Using whiteboards: [DISC 2010], using Pebbles?)
- We considered rendezvous in graphs containing (permanent) faults. Can we solve rendezvous in a dynamic graph?
- Is it possible to solve rendezvous when agents have constant memory? (*Rings*:Yes, *Torus*:Yes [Kranakis et al.] Other topology?)