Chapter 10 File Systems and Mobile Objects

Distributed Computing Group

Mobile Computing

Summer 2002

Overview

- File Systems
- Databases

- Distributed Objects in Ad-Hoc Networks
- Arrow Protocol
- Global Variable in Mobile Ad-Hoc Network

File systems - Motivation

- Goal
  - efficient and transparent access to shared files within a mobile environment while maintaining data consistency
- Problems
  - limited resources of mobile computers (memory, CPU, ...)
  - low bandwidth, variable bandwidth, temporary disconnection
  - high heterogeneity of hardware and software components (no standard PC architecture)
  - wireless network resources and mobile computer are not very reliable
  - standard file systems (e.g. NFS) are very inefficient, almost unusable
- Solutions
  - replication of data (copying, cloning, caching)
  - data collection in advance (hoarding, pre-fetching)

File systems - consistency problems

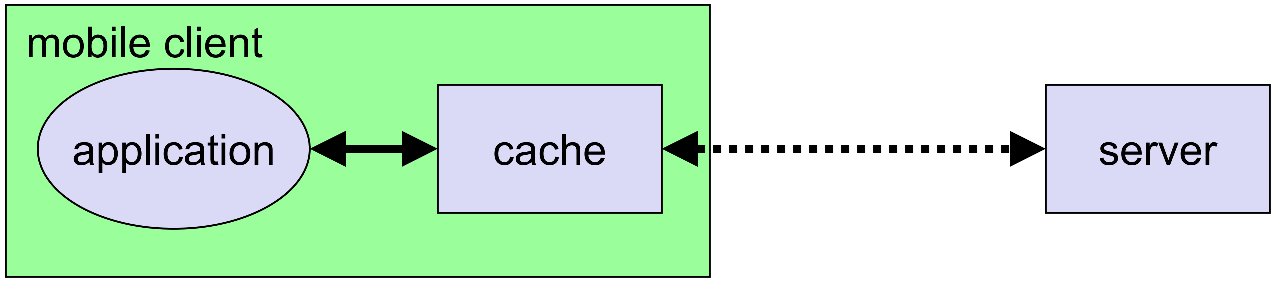
- A central problem of distributed, loosely coupled systems
  - are all views on data the same?
  - how and when should changes be propagated to what users?
- Strong consistency
  - many algorithms offering strong consistency like in database systems (via atomic updates) cannot be used in mobile environments
  - invalidation of data located in caches through a server is very problematic if the mobile computer is currently not connected to the network
- Weak consistency
  - occasional inconsistencies have to be tolerated, but conflict resolution strategies must be applied afterwards to reach consistency again
- Conflict detection
  - content independent: version numbering, time-stamps
  - content dependent: dependency graphs

File system variables

- Client/Server or Peer-to-Peer relations
- Support in the fixed network and/or mobile computers
- One file system (or namespace) or several file systems
- Transparency
  - hide the mobility support, applications on mobile computers should not notice the mobility
  - user should not notice additional mechanisms needed
- Optimistic or pessimistic consistency model
- Caching and Pre-fetching
  - bytes, paragraphs, single files, directories, subtrees, partitions, ...
  - permanent or only at certain points in time
- Data management
- Conflict solving

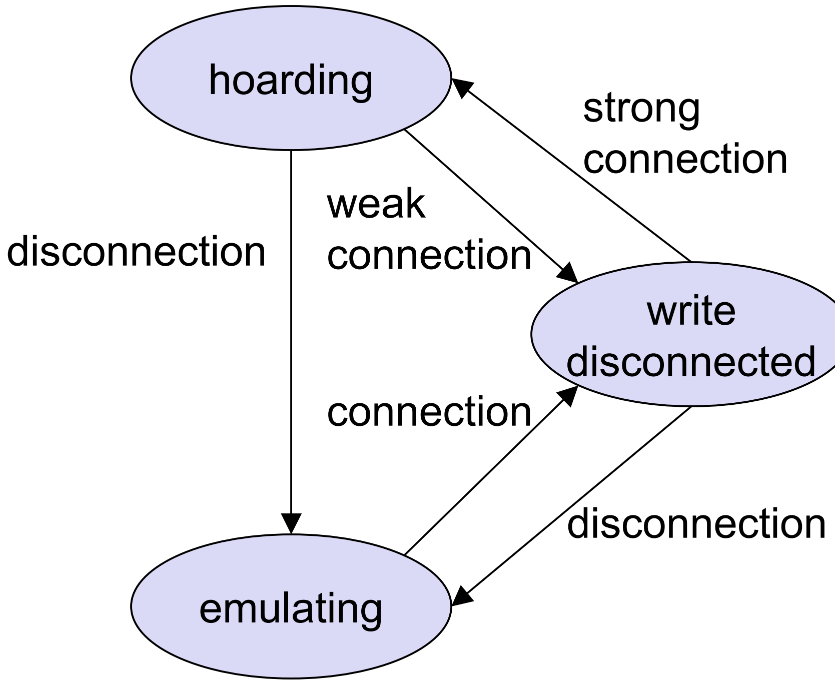
Coda

- Application transparent extensions of client and server
  - changes in the cache manager of a client
  - applications use cache replicates of files
  - extensive, transparent collection of data in advance for possible future use („hoarding“)
- Consistency
  - system keeps a record of changes in files and compares files after reconnection
  - if different users have changed the same file a manual reintegration of the file into the system is necessary
  - optimistic approach, coarse-grained (file size)

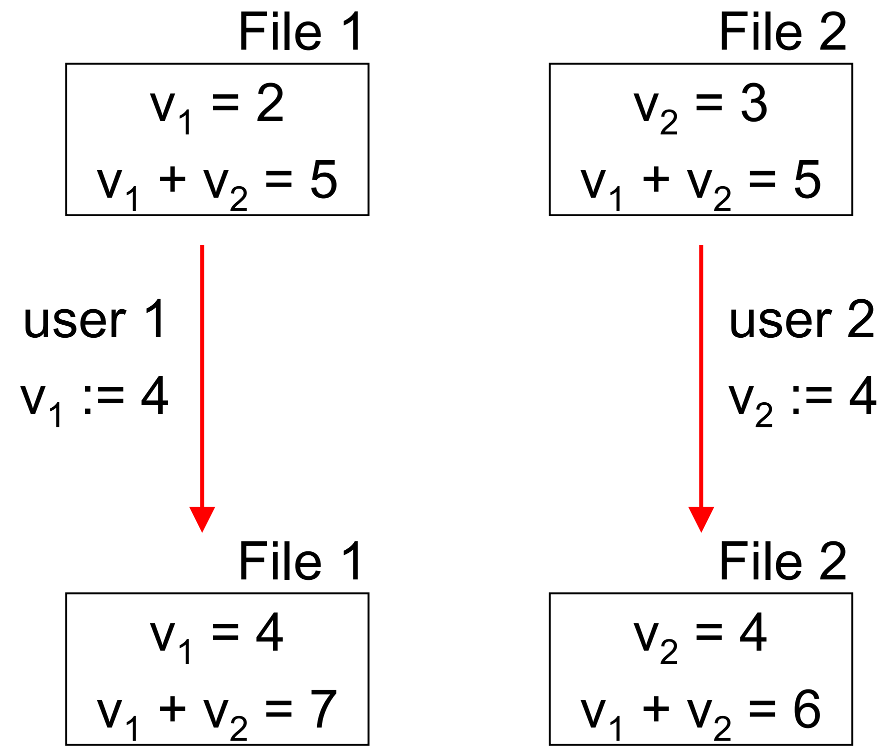


Coda – some functionality

- Hoarding
  - user can pre-determine a file list with priorities
  - contents of the cache determined by the list and LRU strategy (Least Recently Used)
  - explicit pre-fetching possible
  - periodic updating
- Comparison of files
  - asynchronous, background
  - system weighs speed of updating against minimization of network traffic
- Cache misses
  - modeling of user patience: how long can a user wait for data without an error message?
  - function of file size and bandwidth
- States of a client



Coda Transaction Mode



- File check-in is not a problem
- Solution: transaction mode as an option in Coda

Little Work

- Another extension of AFS
- Only changes in the cache manager of the client
- Connection modes:

	Connected	**Partially Connected**	**Fetch only**	**Disconnected**
Method	normal	delayed write to the server	optimistic replication of files	abort at cache miss
Network requirements	continuous high bandwidth	continuous bandwidth	connection on demand	none
Connection Example	Office, WLAN	packet radio	cellular systems (e.g., GSM) with costs per call	independent

File systems – more examples

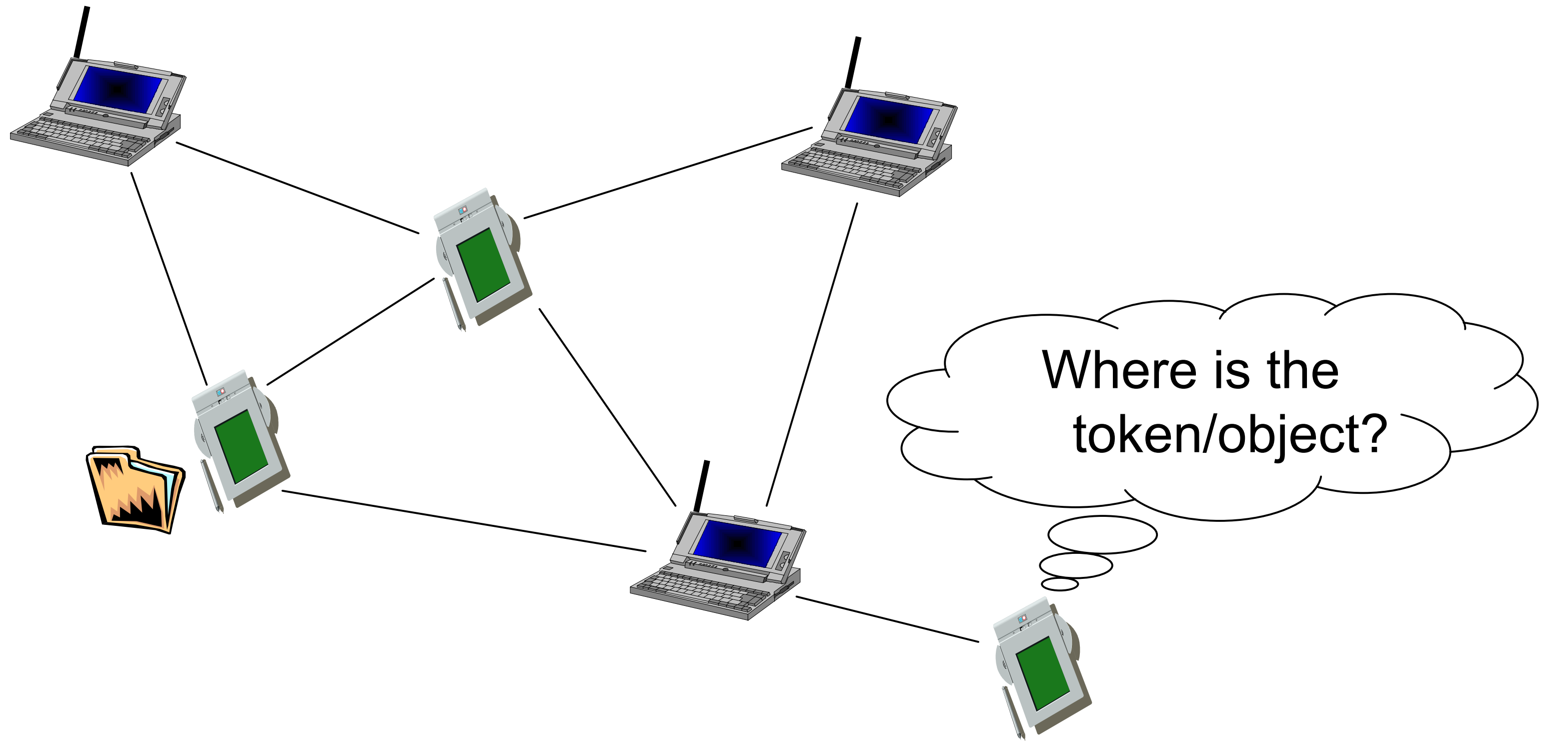
- Ficus
  - not a client/server approach
  - use of „gossip“ protocols: a mobile computer does not necessarily need to have direct connection to a server, with the help of other mobile computers updates can be propagated through the network
  - optimistic approach based on replicates
  - detection of write conflicts, conflict resolution on directory level

- MIo-NFS (Mobile Integration of NFS)
  - NFS extension
  - pessimistic approach: only token holder can write
  - Three modes: connected, loosely connected, disconnected

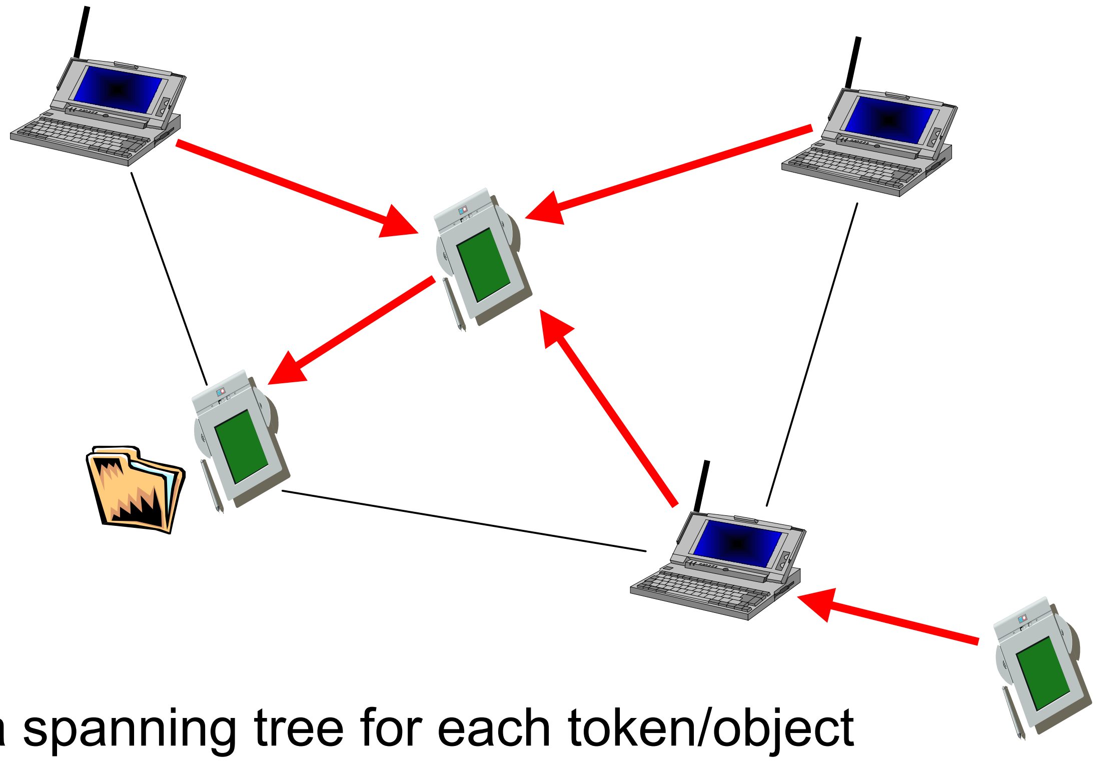
Database systems in mobile environments

- Request processing
  - power conserving, location dependent, cost efficient
- Replication management
  - similar to file systems
- Location management
  - tracking of mobile users to provide replicated or location dependent data in time at the right place (minimize access delays)
  - example: with the help of the VLR (Visitor Location Register) in GSM a mobile user can find a local towing service
- Transaction processing
  - “mobile” transactions can not necessarily rely on the same models as transactions over fixed networks (ACID: atomicity, consistency, isolation, durability)
  - therefore models for “weak” transaction

Mobile Objects in Ad-Hoc Networks

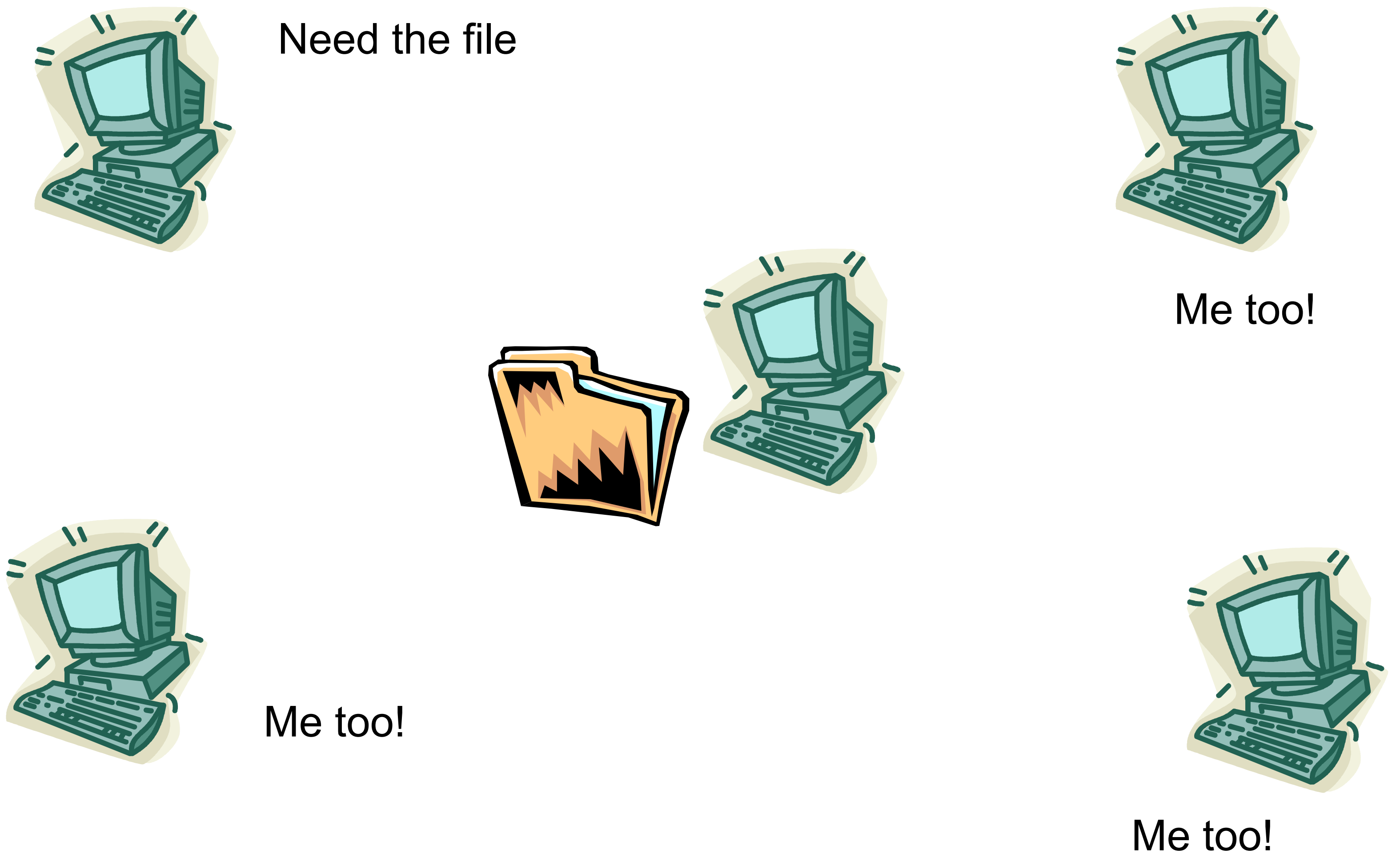


The Arrow Protocol

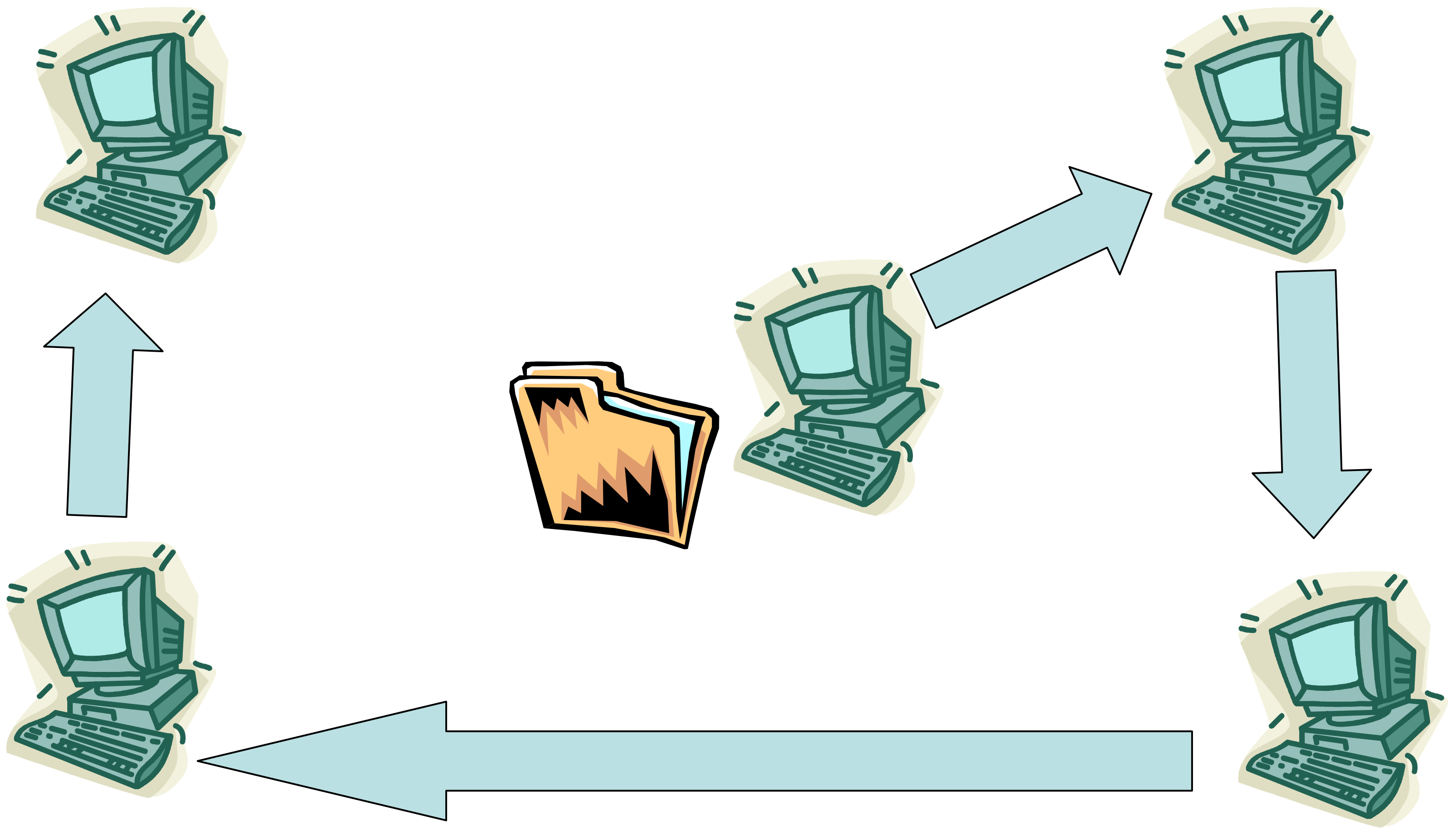


- Build a spanning tree for each token/object
- Links of spanning tree are directed (“Arrows”) that point towards the node that currently has the token/object

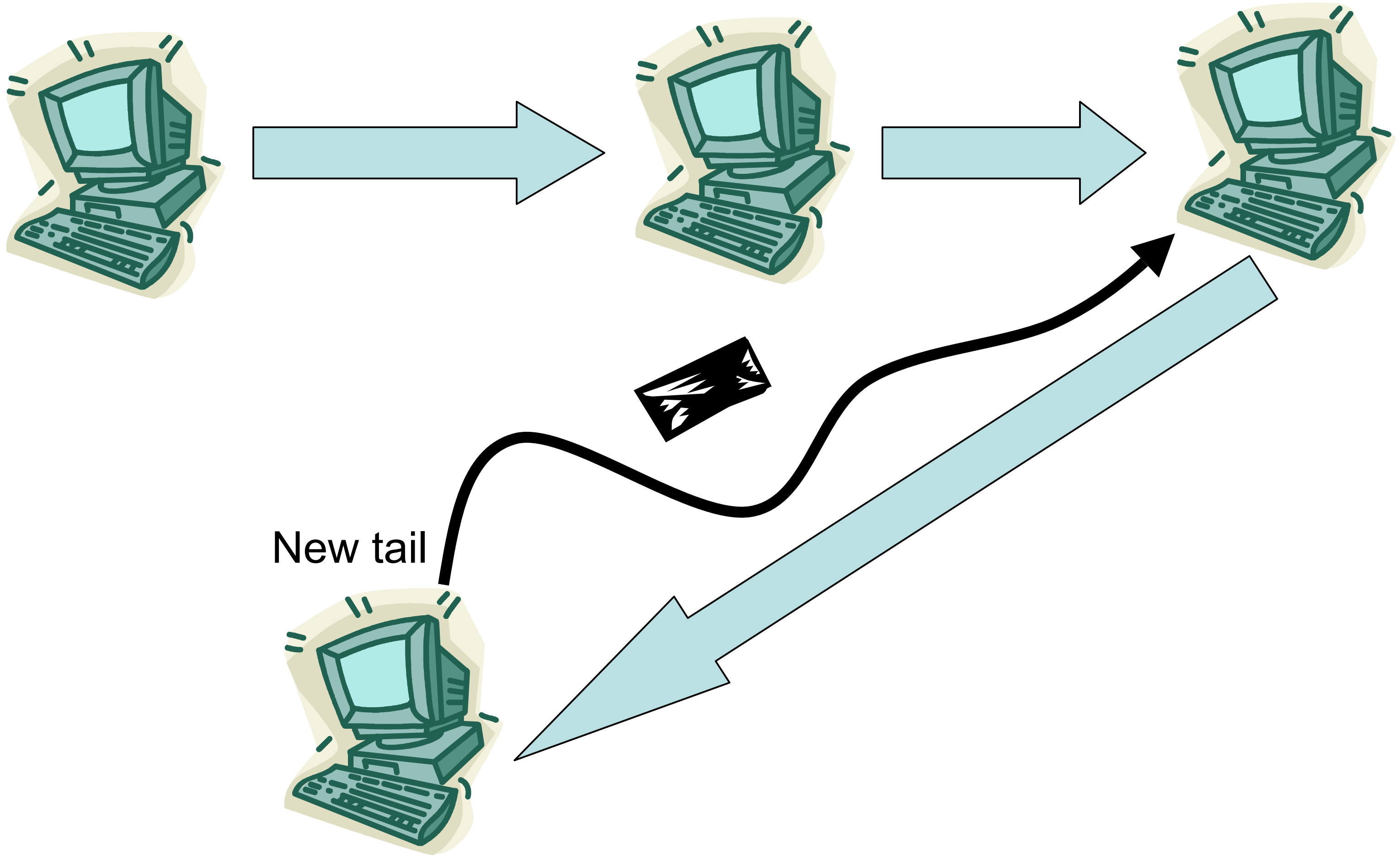
Synchronize Access to Mobile Object



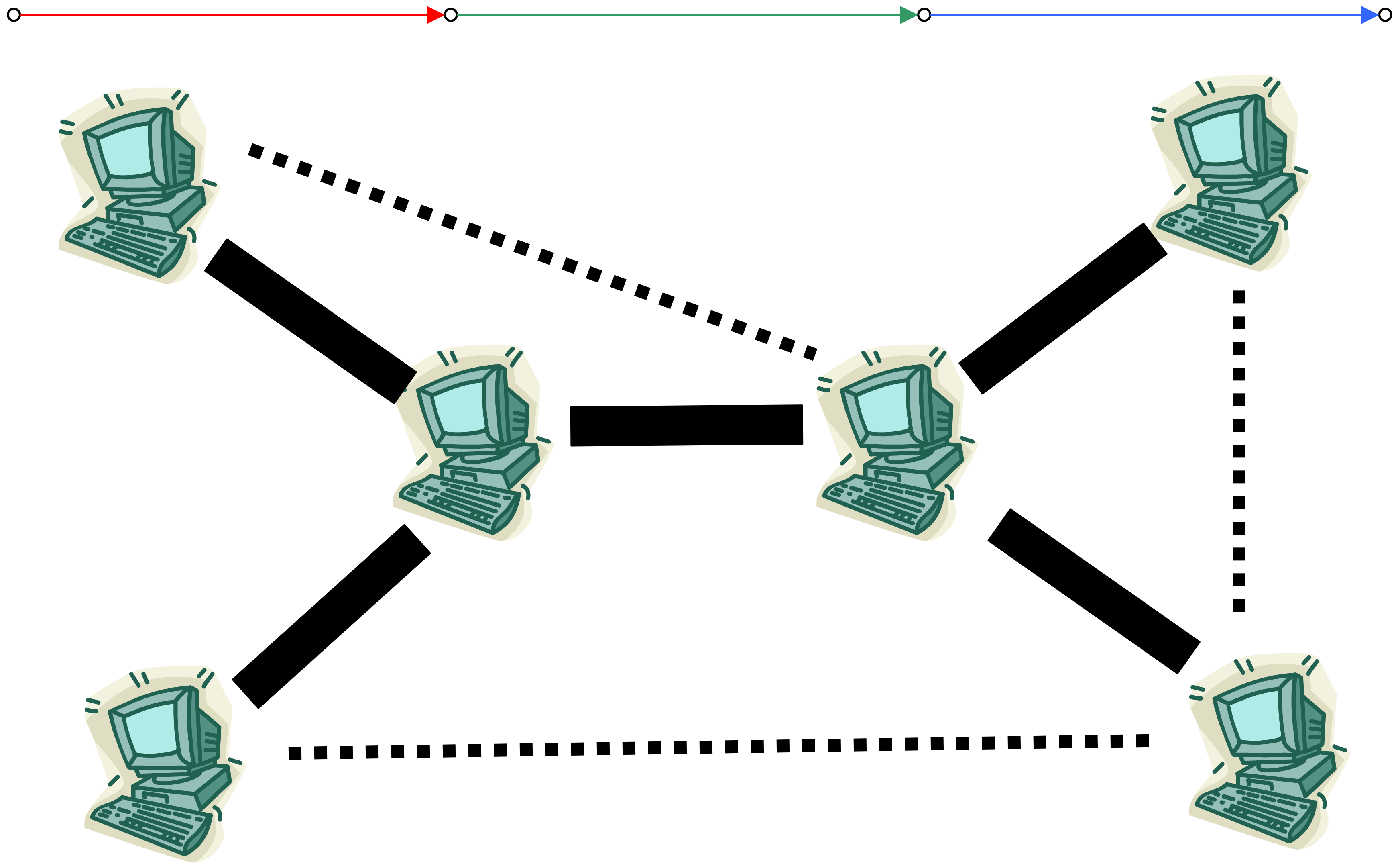
Requests are queued and object/token passed along path



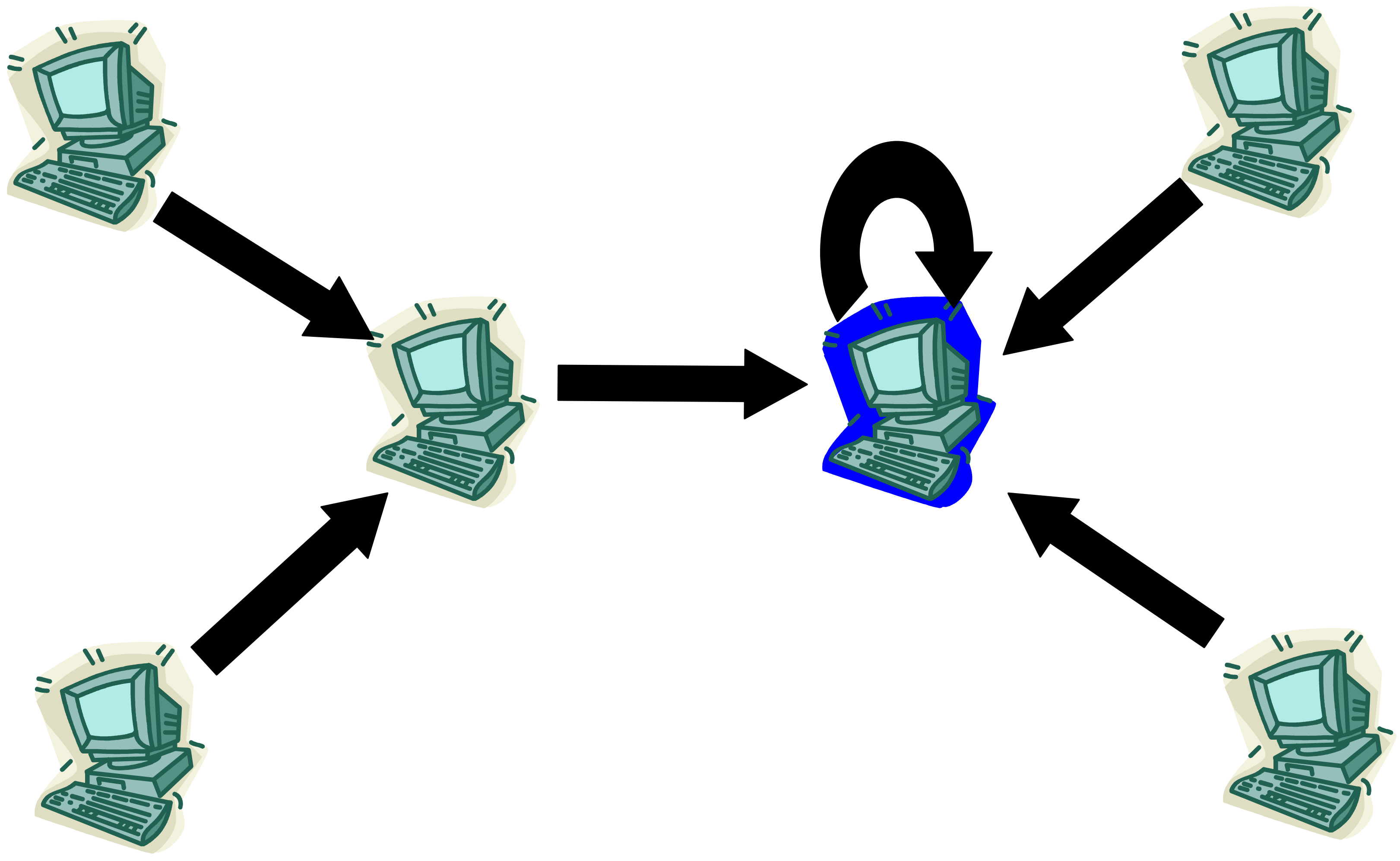
Join Queue = Inform Tail



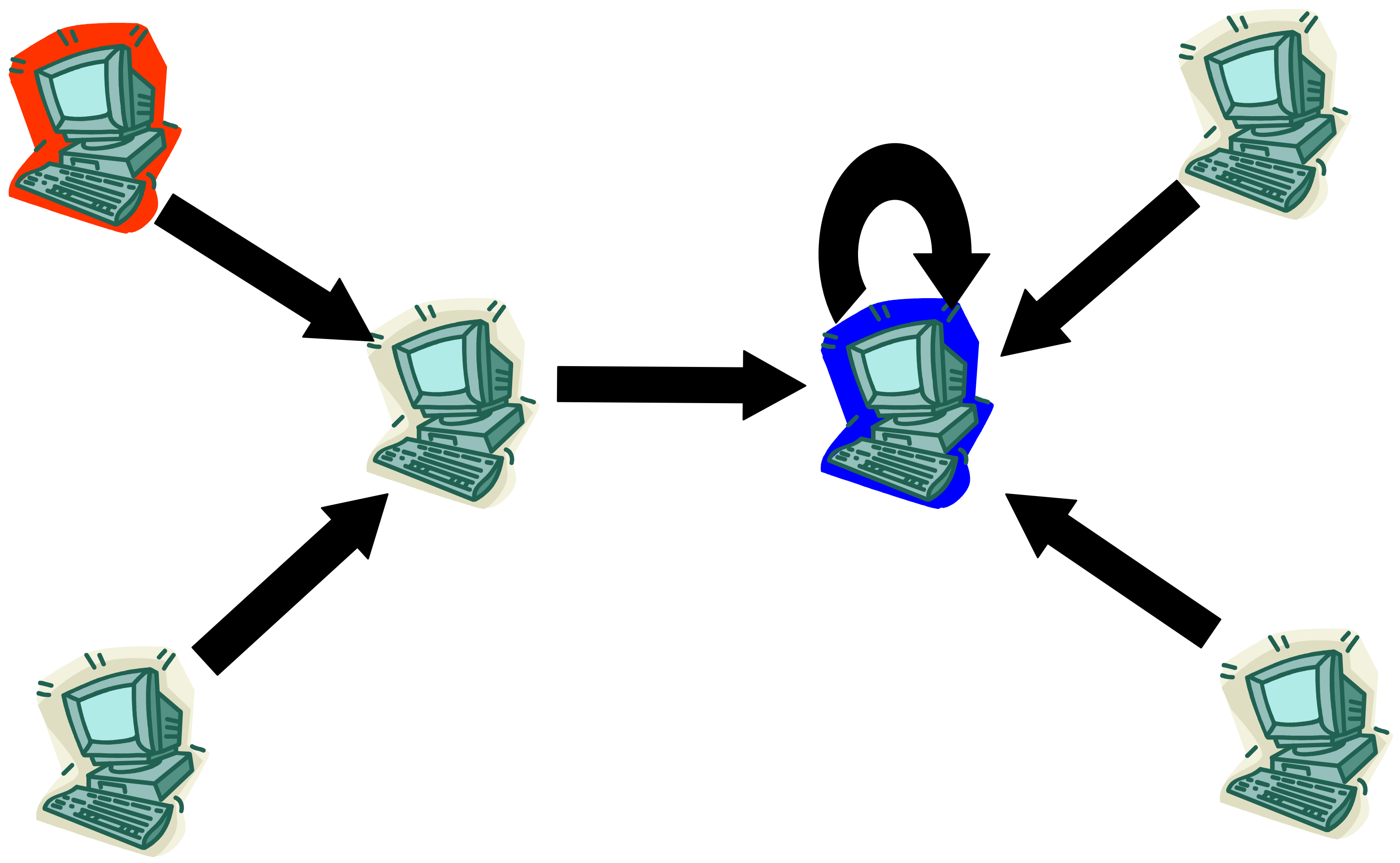
Initialization: Spanning Tree



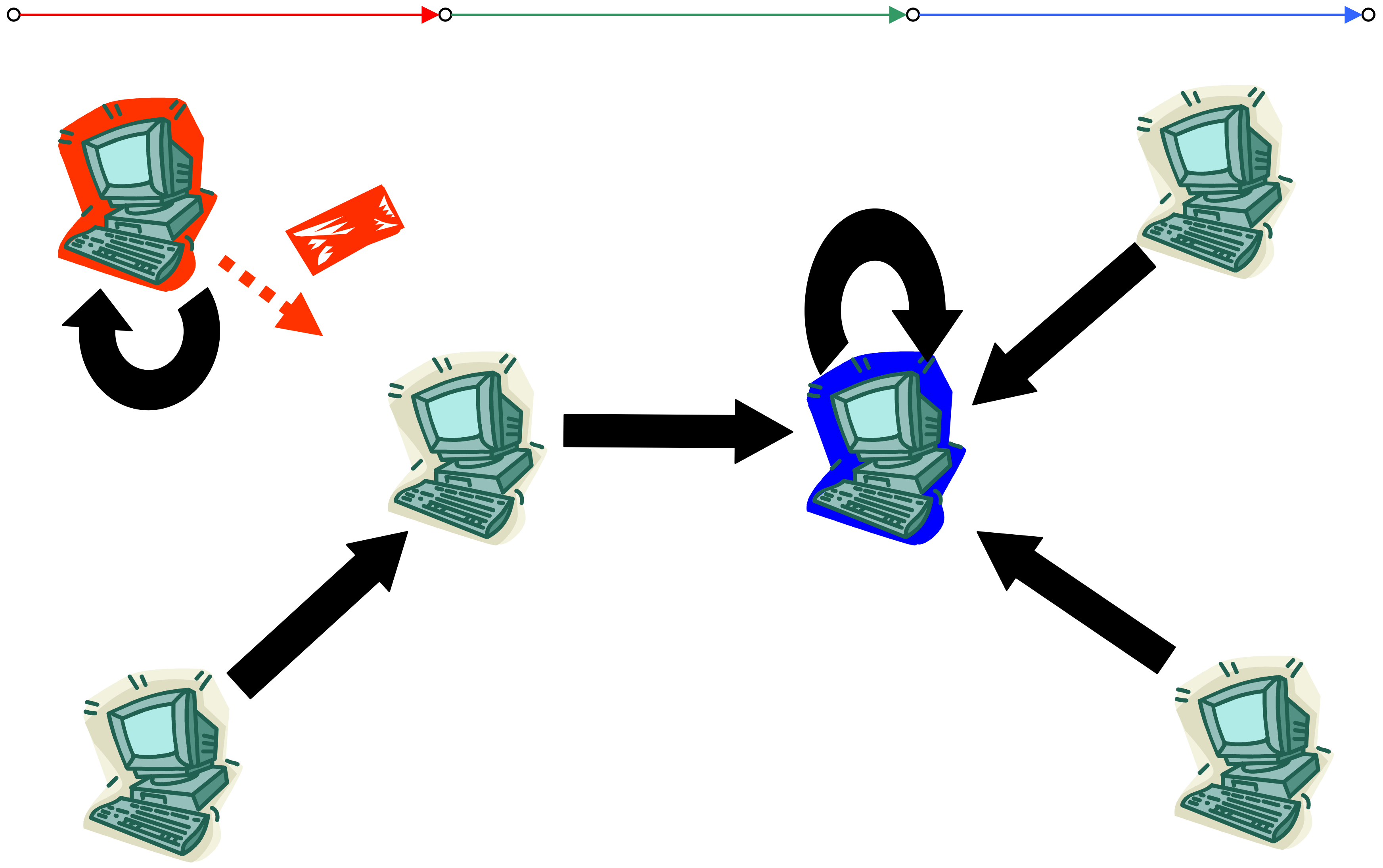
Initialization: Arrows



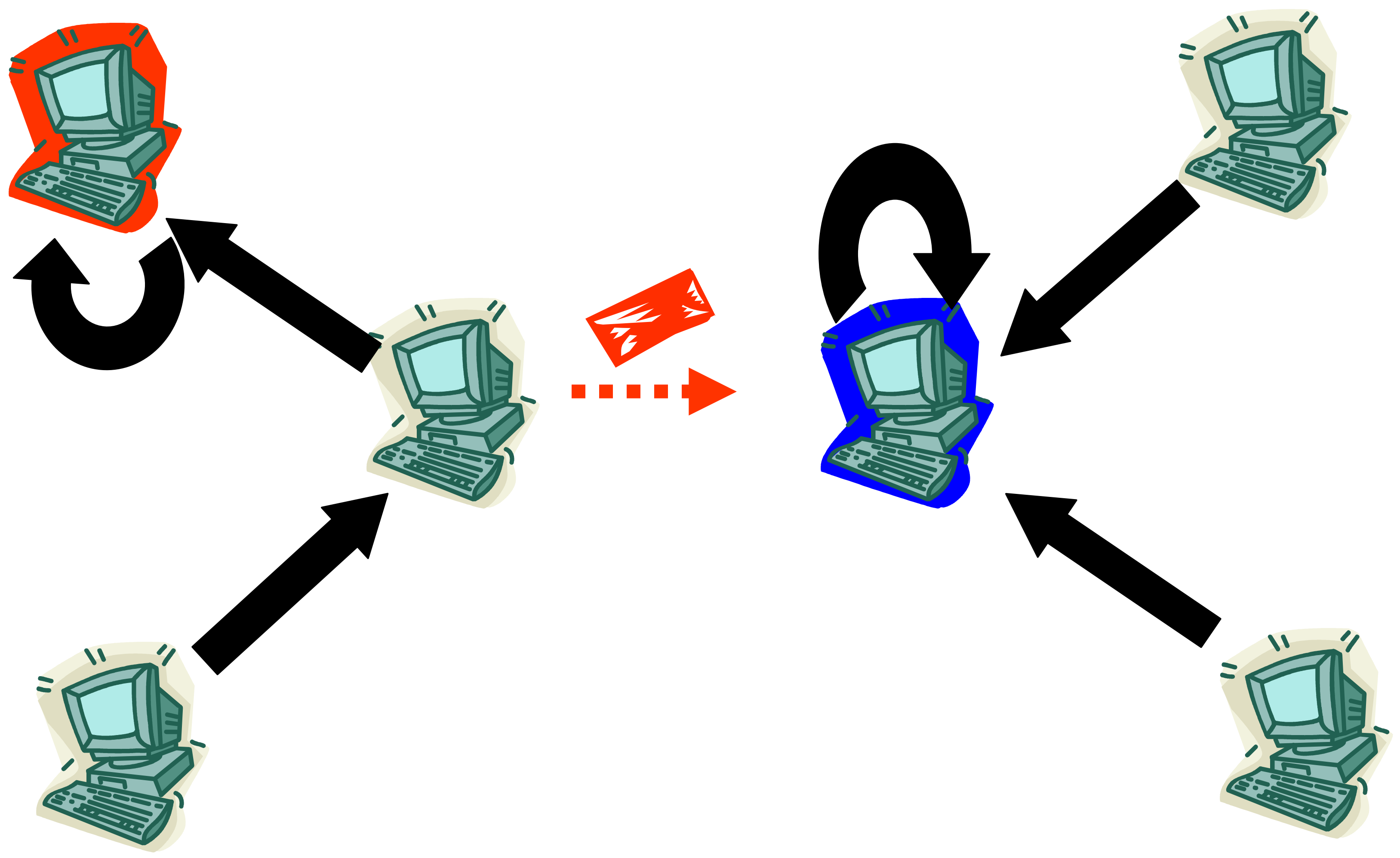
New Request



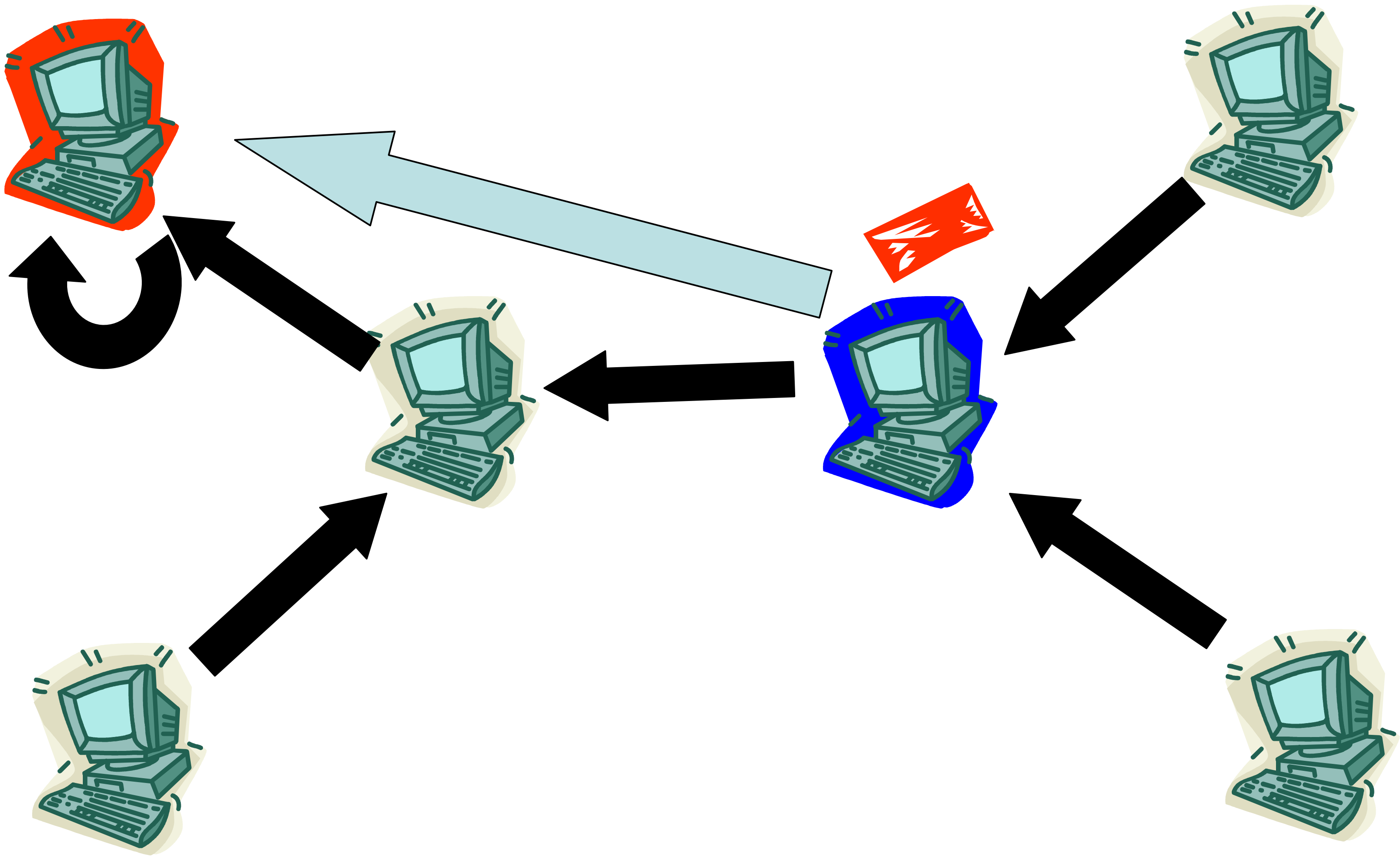
Path Reversal



Path Reversal



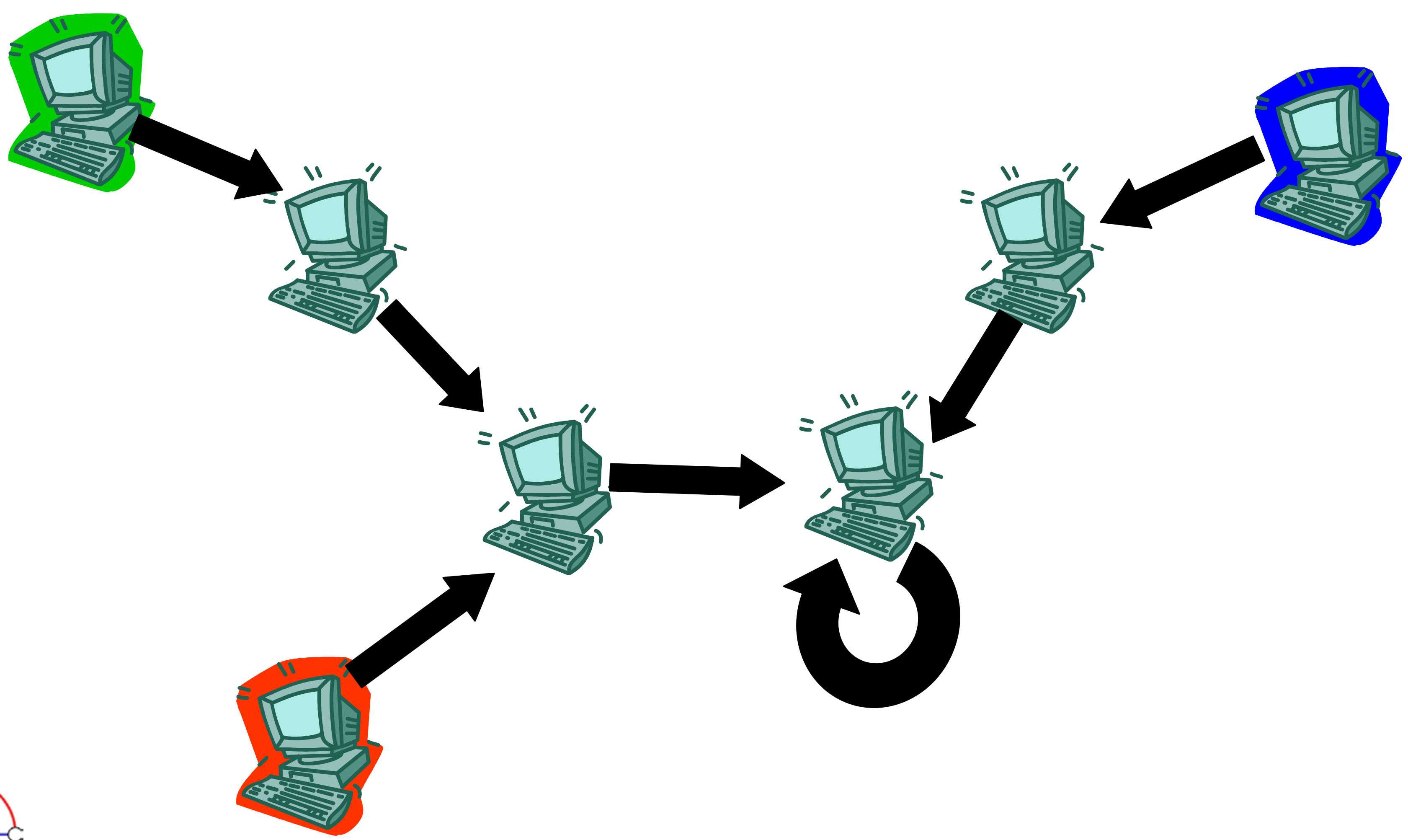
Path Reversal



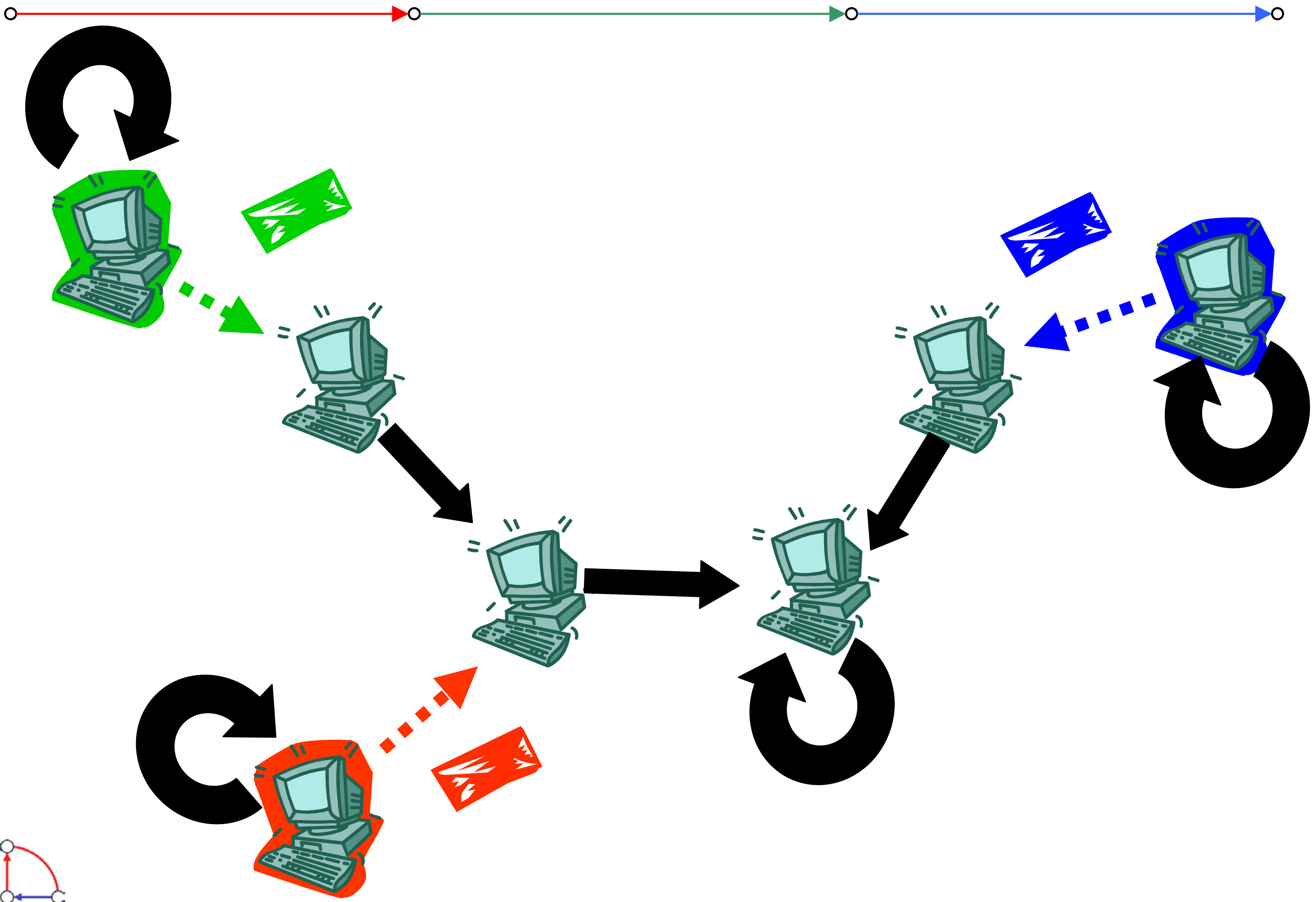
Efficiency of Arrow Protocol

- Definition: Let the latency of a request be the number of hops the request takes until it arrives at the token (or the end of the queue).
- Theorem: The latency of a request is bounded by the diameter of the spanning tree.
- What if we have r simultaneous requests? We hope that most requests will be queued locally.
- Definition: The cost of r simultaneous requests is the sum of the latencies of the r requests.
- Theorem: The competitive ratio of r simultaneous requests is log r. There is an almost matching lower bound of log r / loglog r.

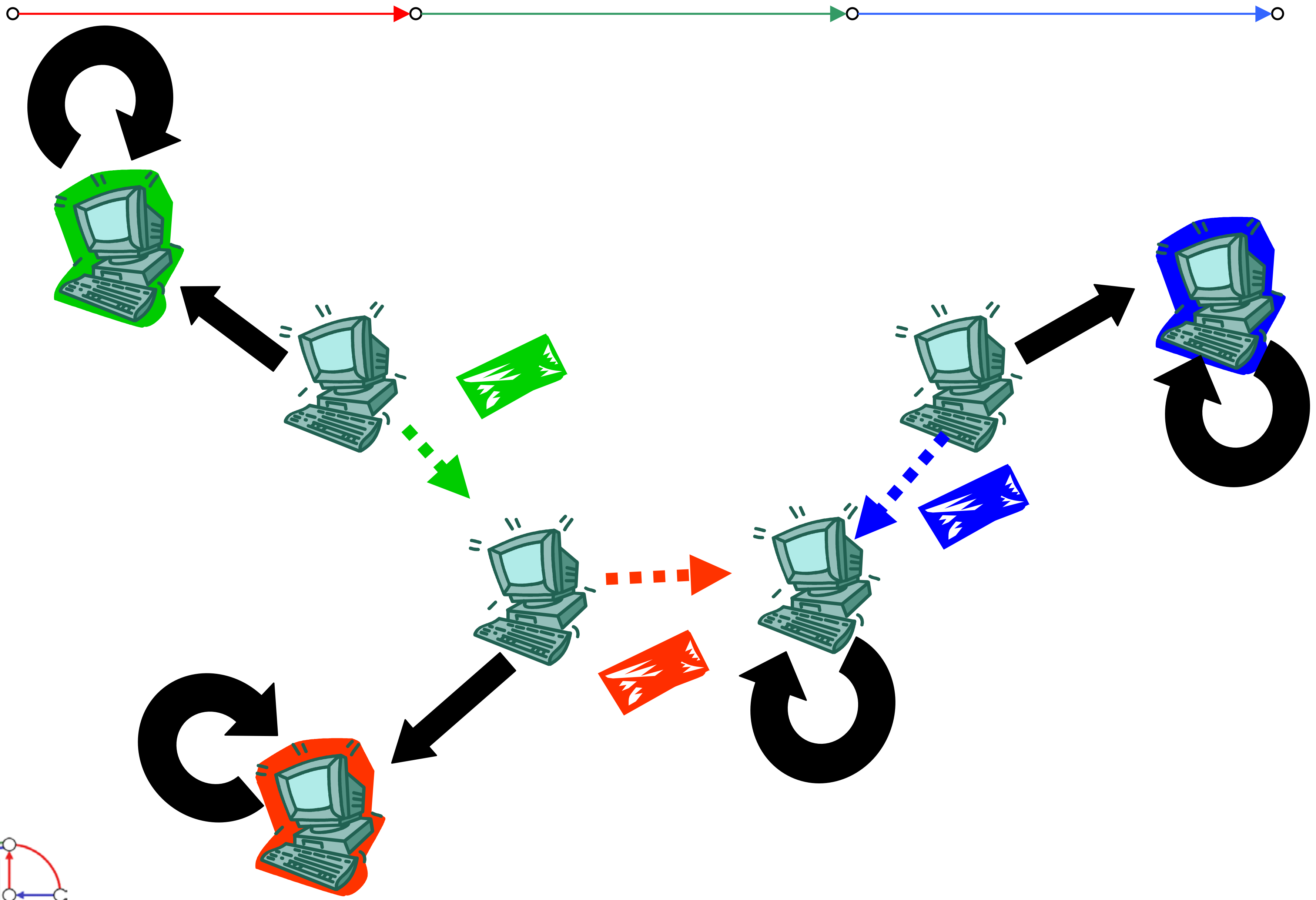
Example for Concurrent Requests



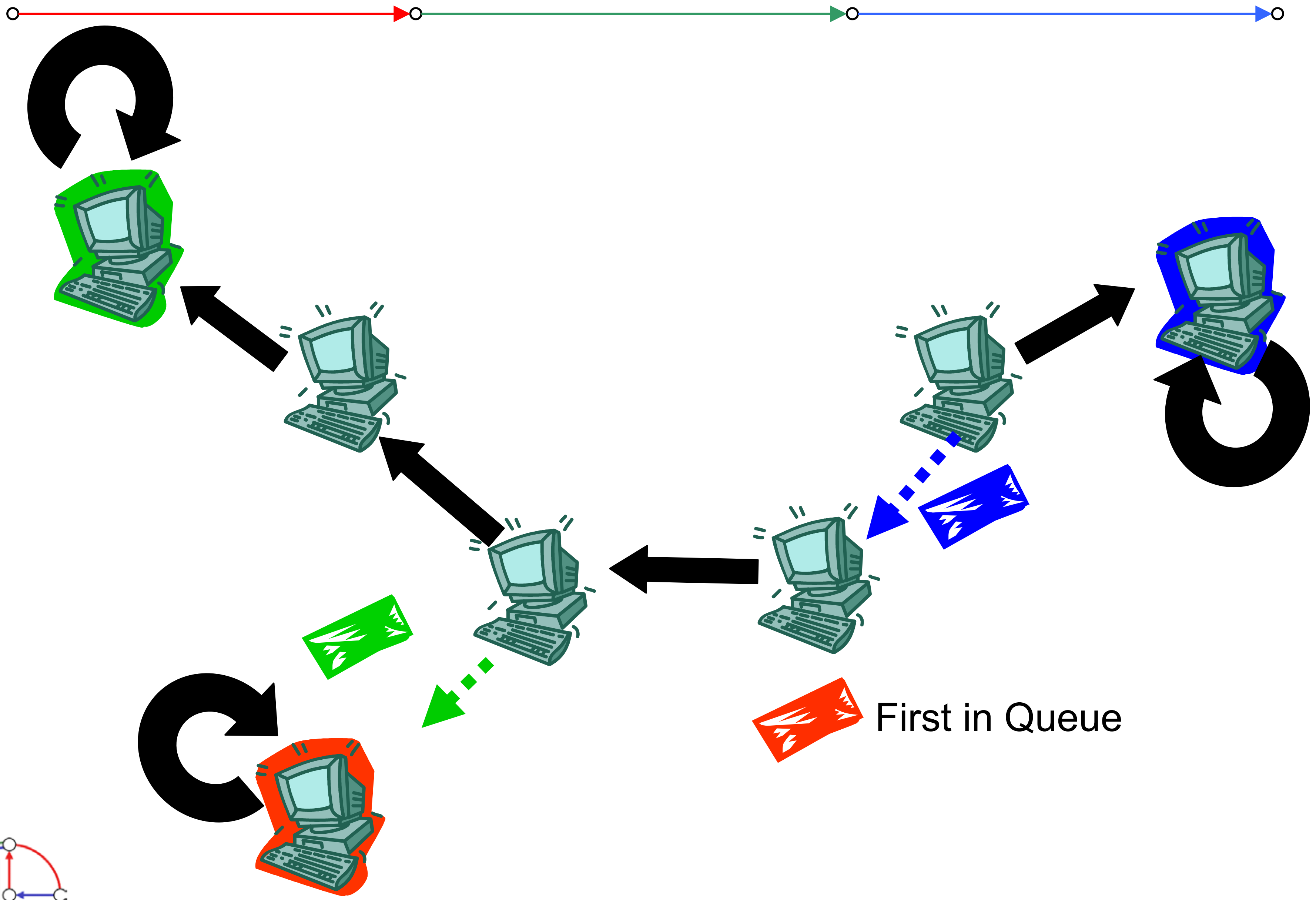
Example for Concurrent Requests



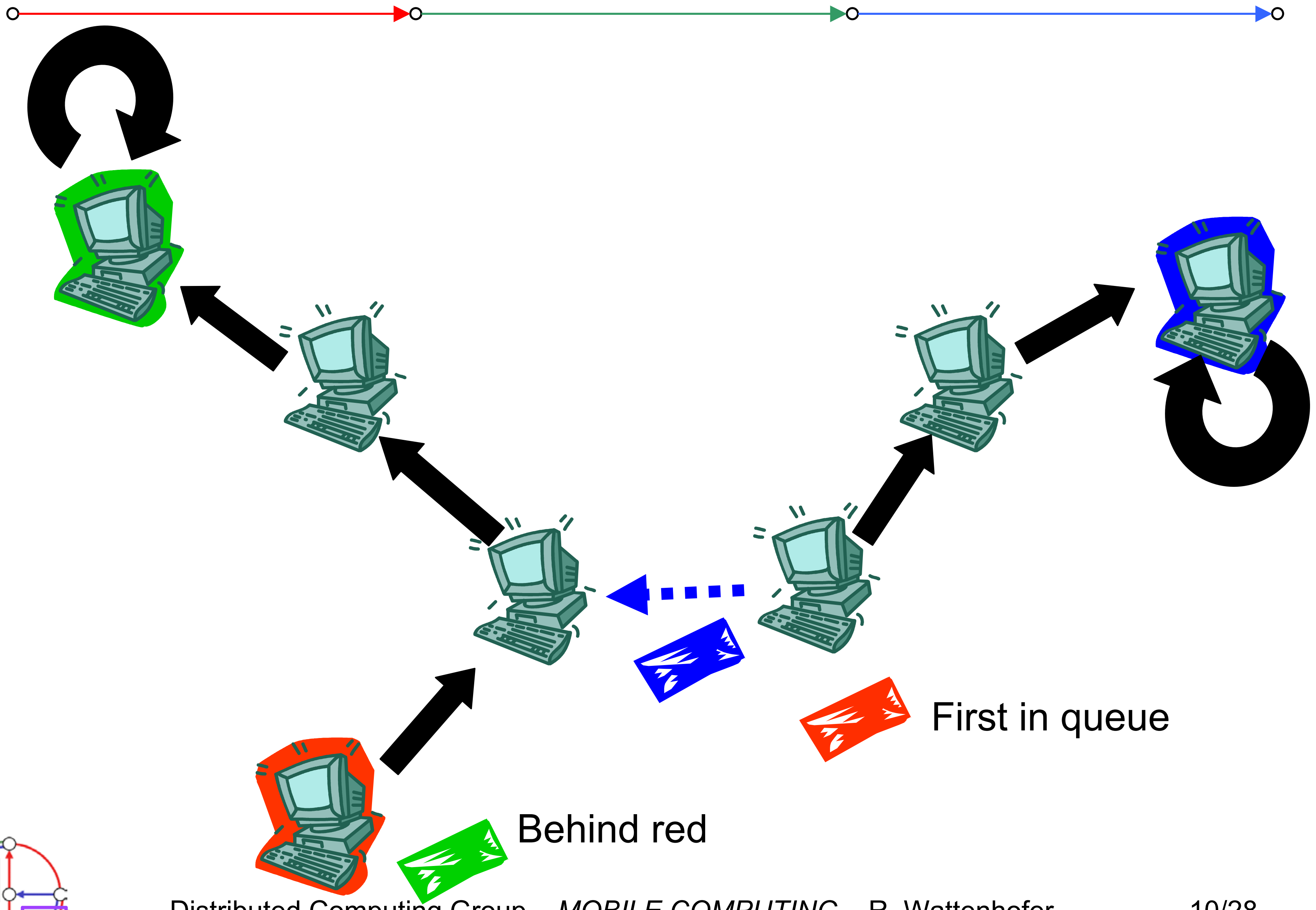
Example for Concurrent Requests



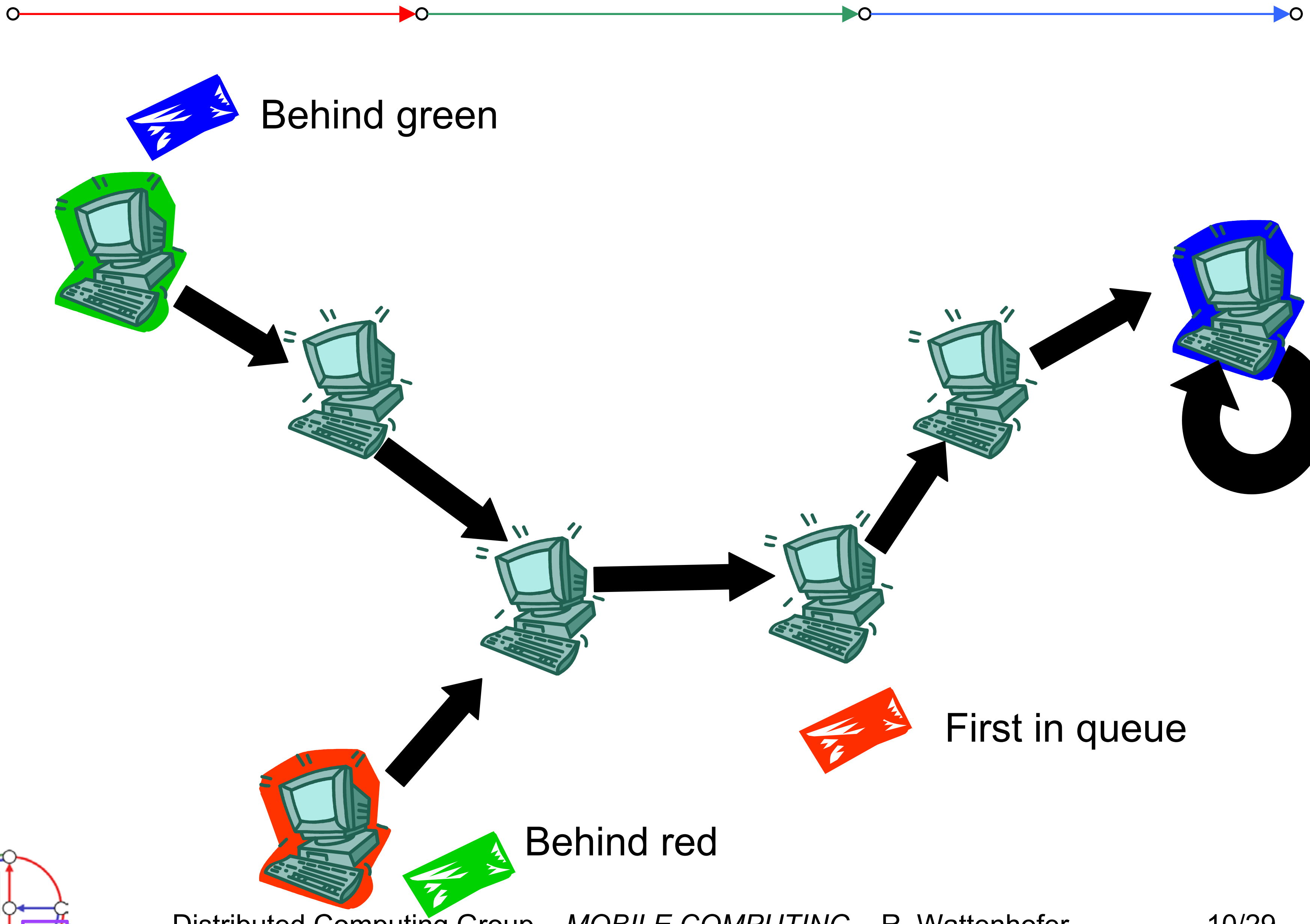
Example for Concurrent Requests



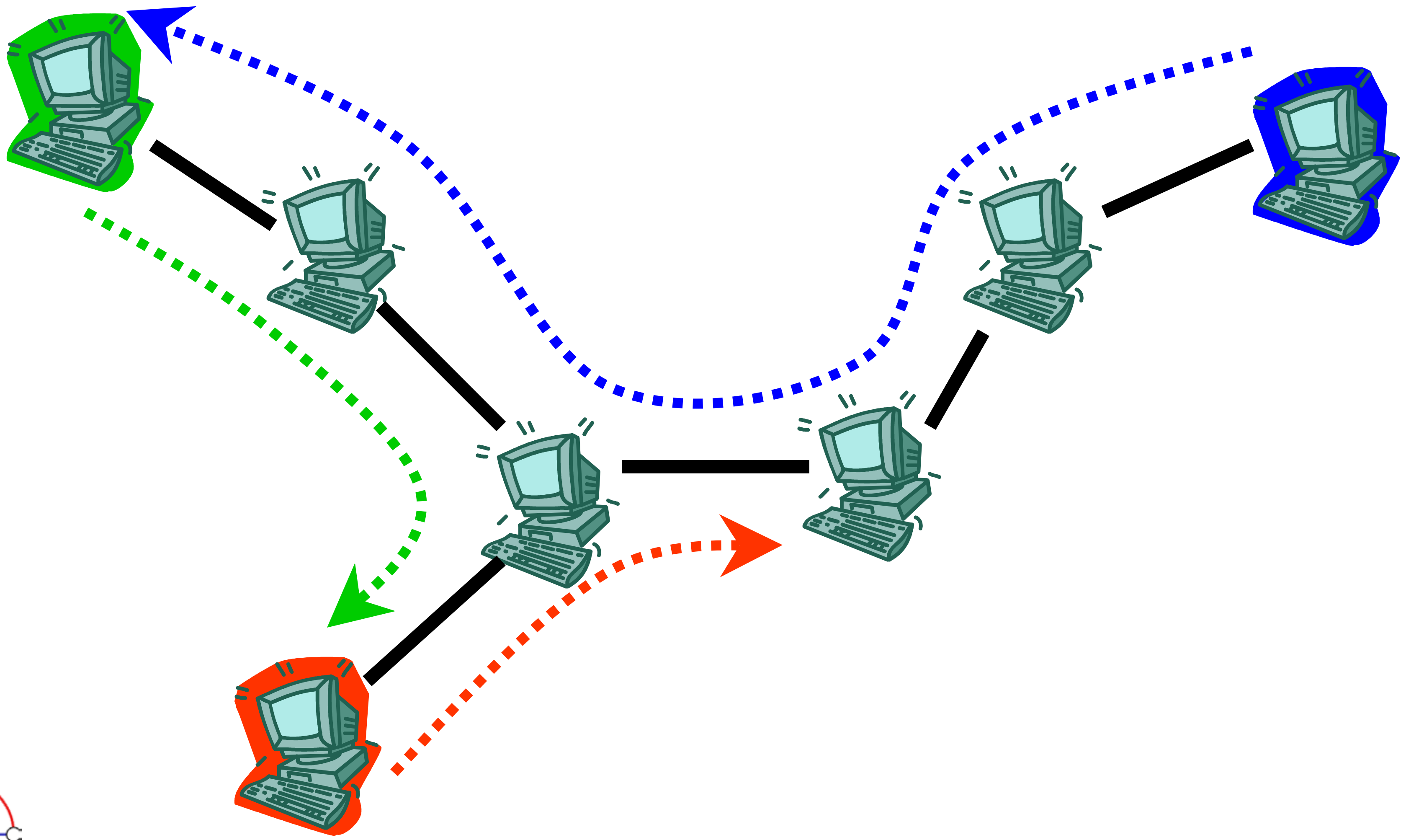
Example for Concurrent Requests



Example for Concurrent Requests



Paths taken by requests



Roadmap of proof of log r competitivity

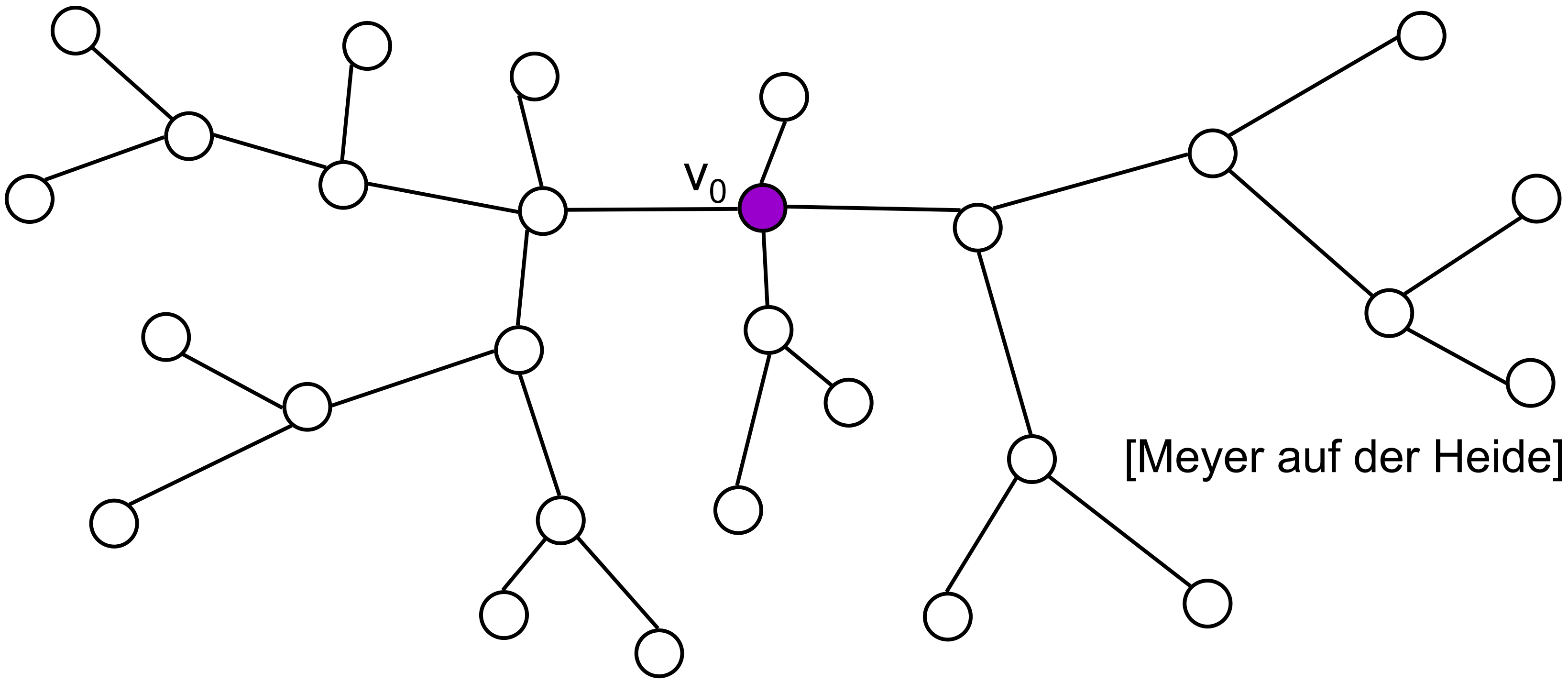
- Upper bound on Cost of Arrow: Nearest Neighbor characterization of order of queuing
- The nearest neighbor TSP heuristic is log r competitive.
- Lower bound on cost for optimal offline algorithm
- On the other hand, there is a worst-case example whose cost is log r / loglog r higher than the optimal offline cost.
- Thus, the competitive ratio is almost tight.
- Open Problem: Dynamic analysis of arrow protocol.

Global Variable in Mobile Ad-Hoc Network

- Application: Sequence of read / write requests from mobile node to global object. Each processor decides solely based on its local knowledge.
- Idea: Use a variant of the arrow protocol to find a copy of the object and replicate the object with each read. A write should then invalidate all replicas.
- Node v writes to variable x: Node v creates (or updates) replica of x in v, and invalidates all other replicas.
- Node v reads variable x: Node *v* reads the closest replica of *x* and creates copies in every node of the tree on the path back to *v*.

Example and Analysis

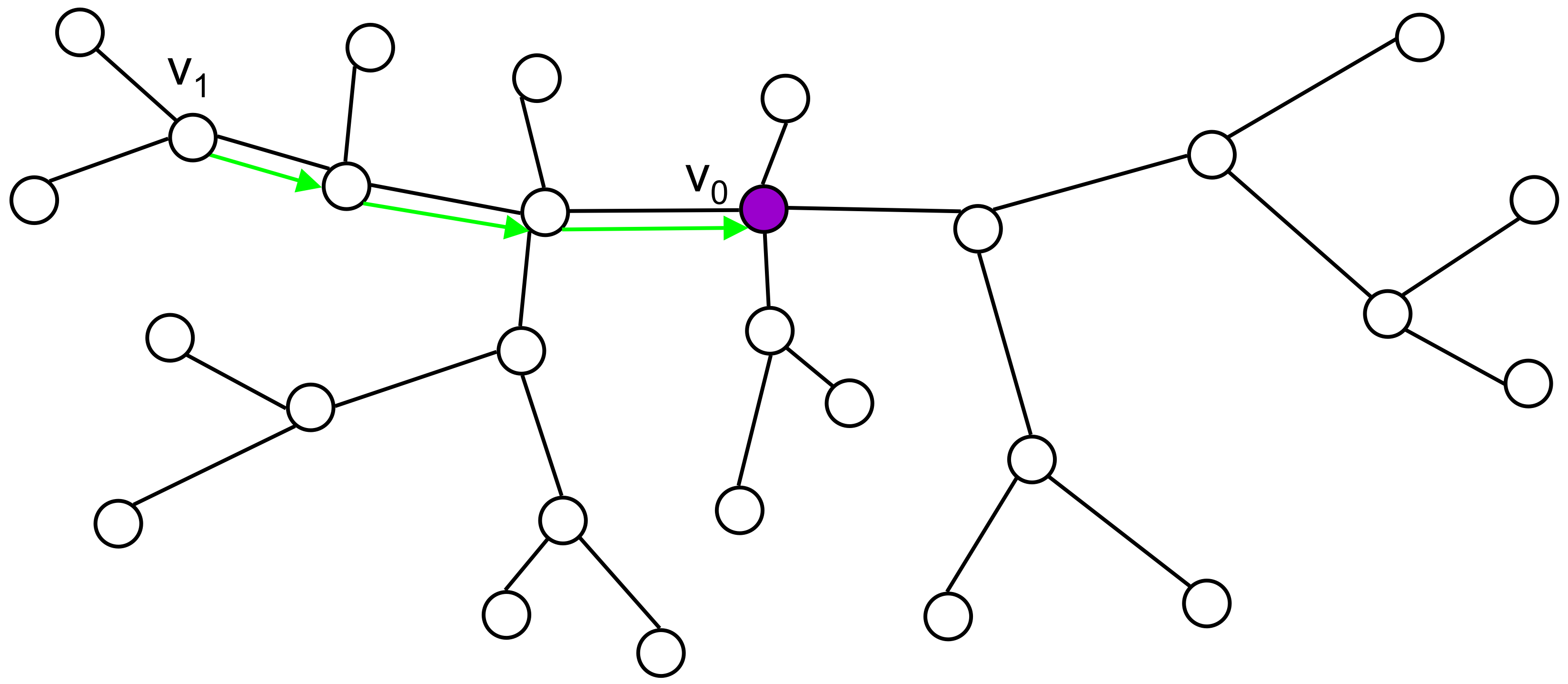
Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



[Meyer auf der Heide]

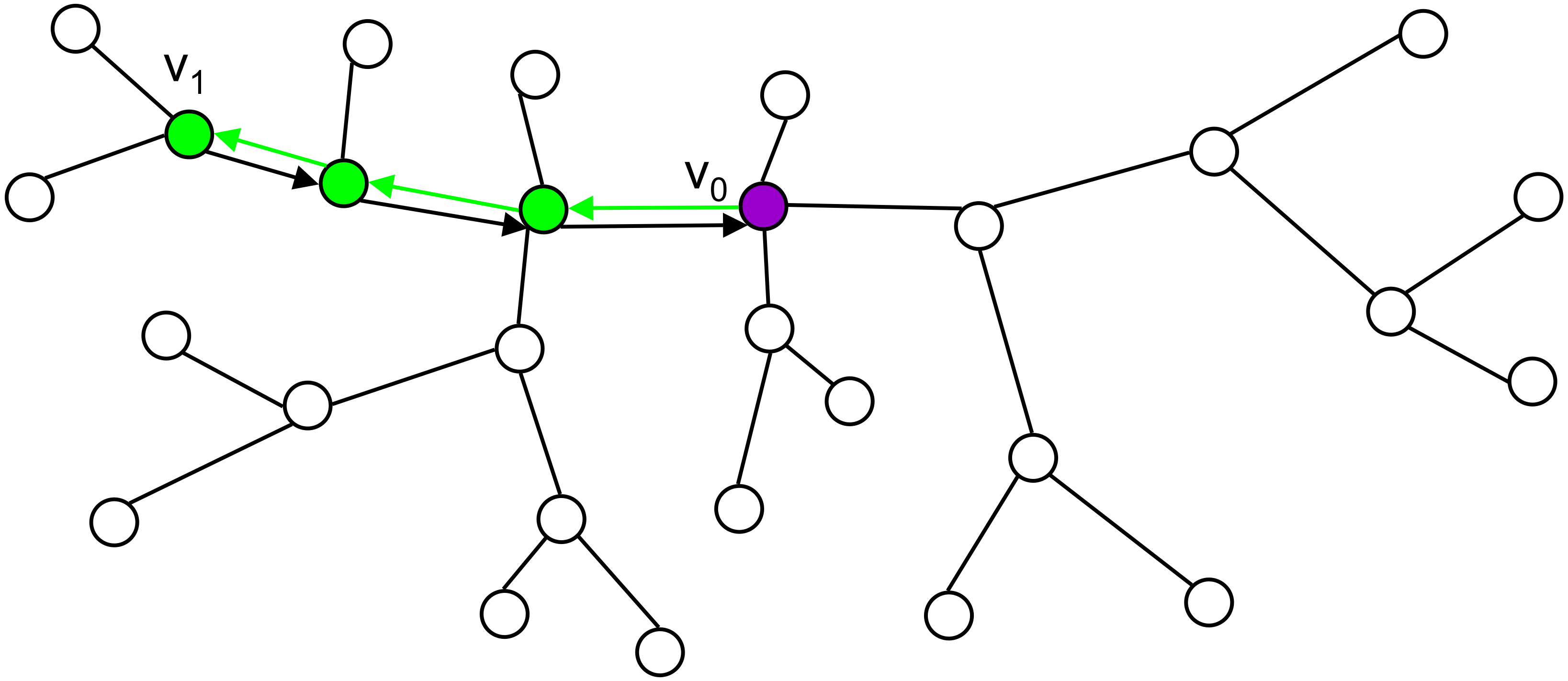
Example and Analysis

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



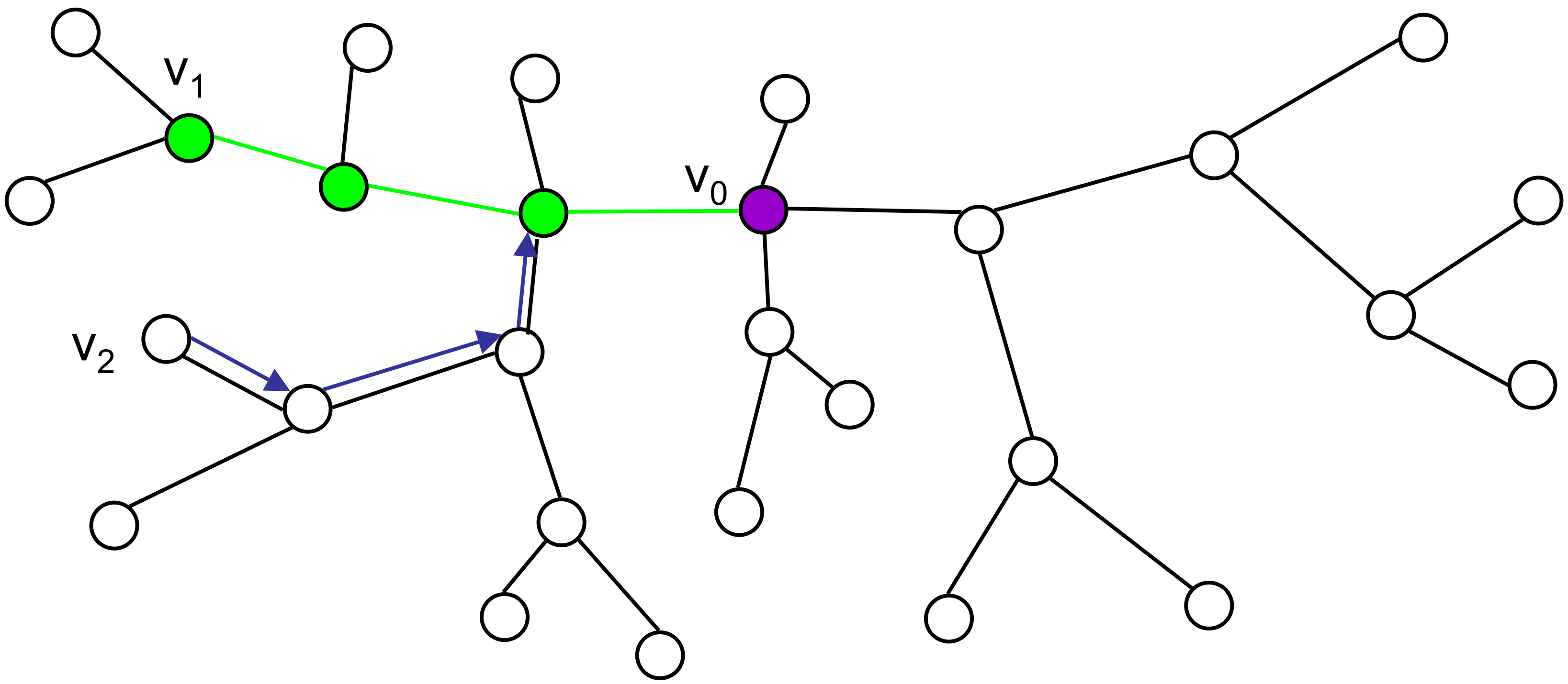
Example and Analysis

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



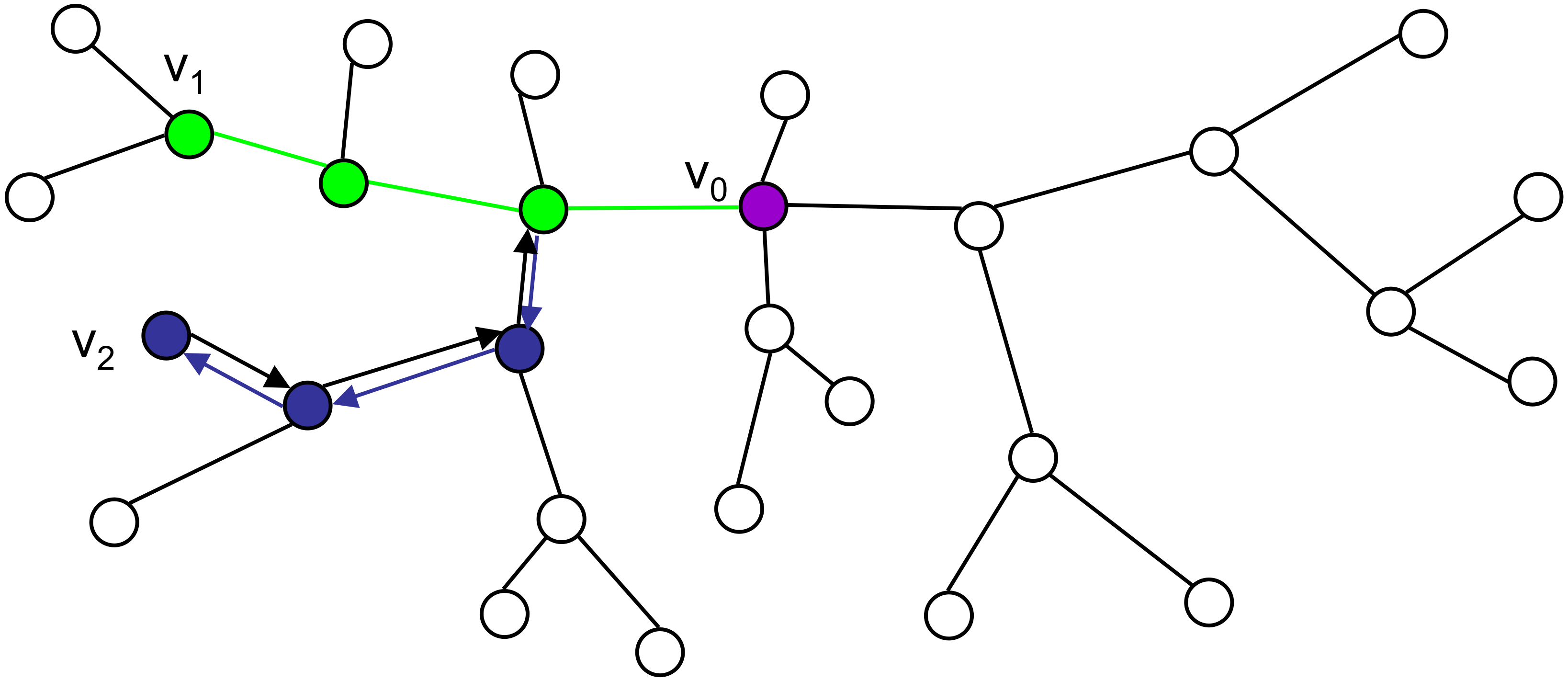
Example and Analysis

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



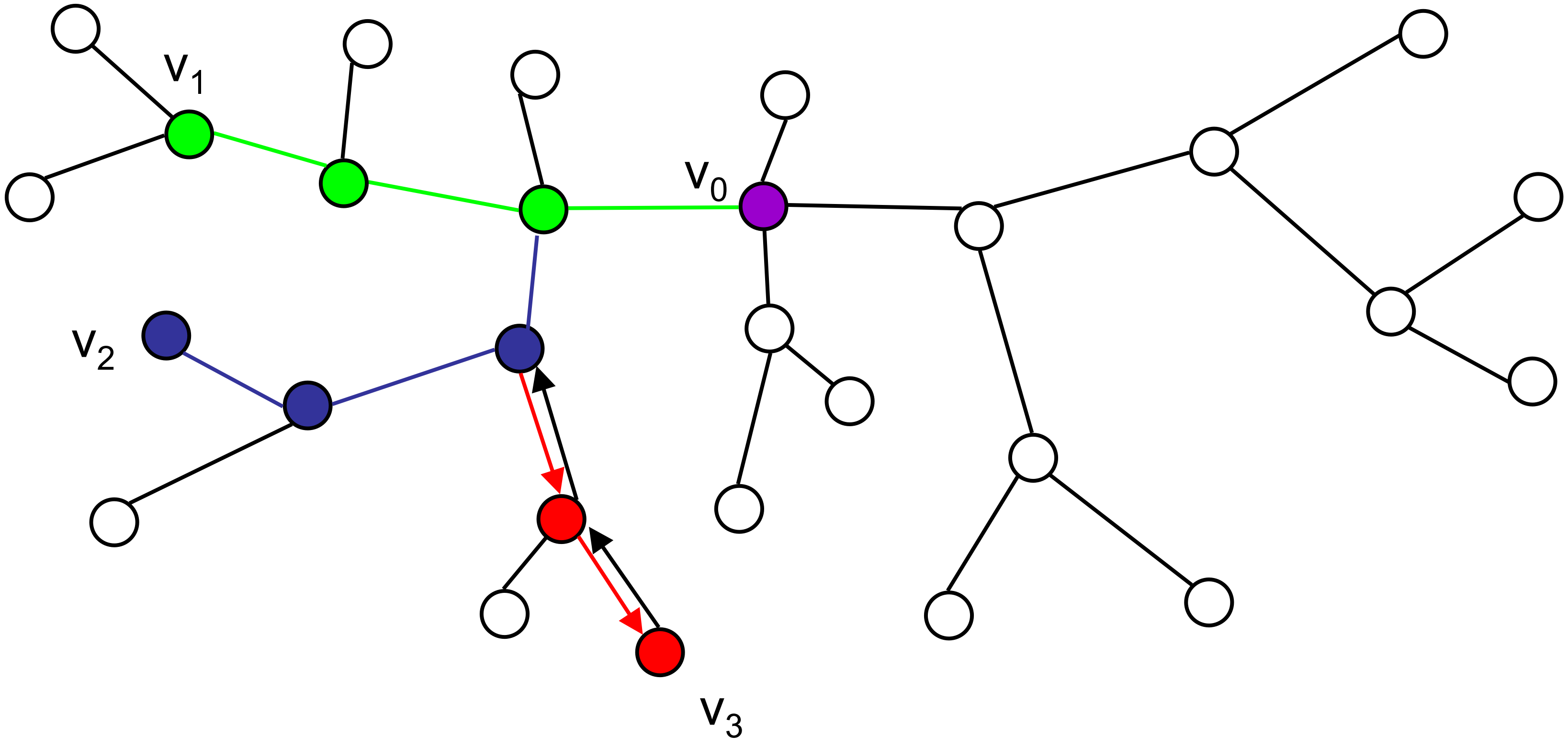
Example and Analysis

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



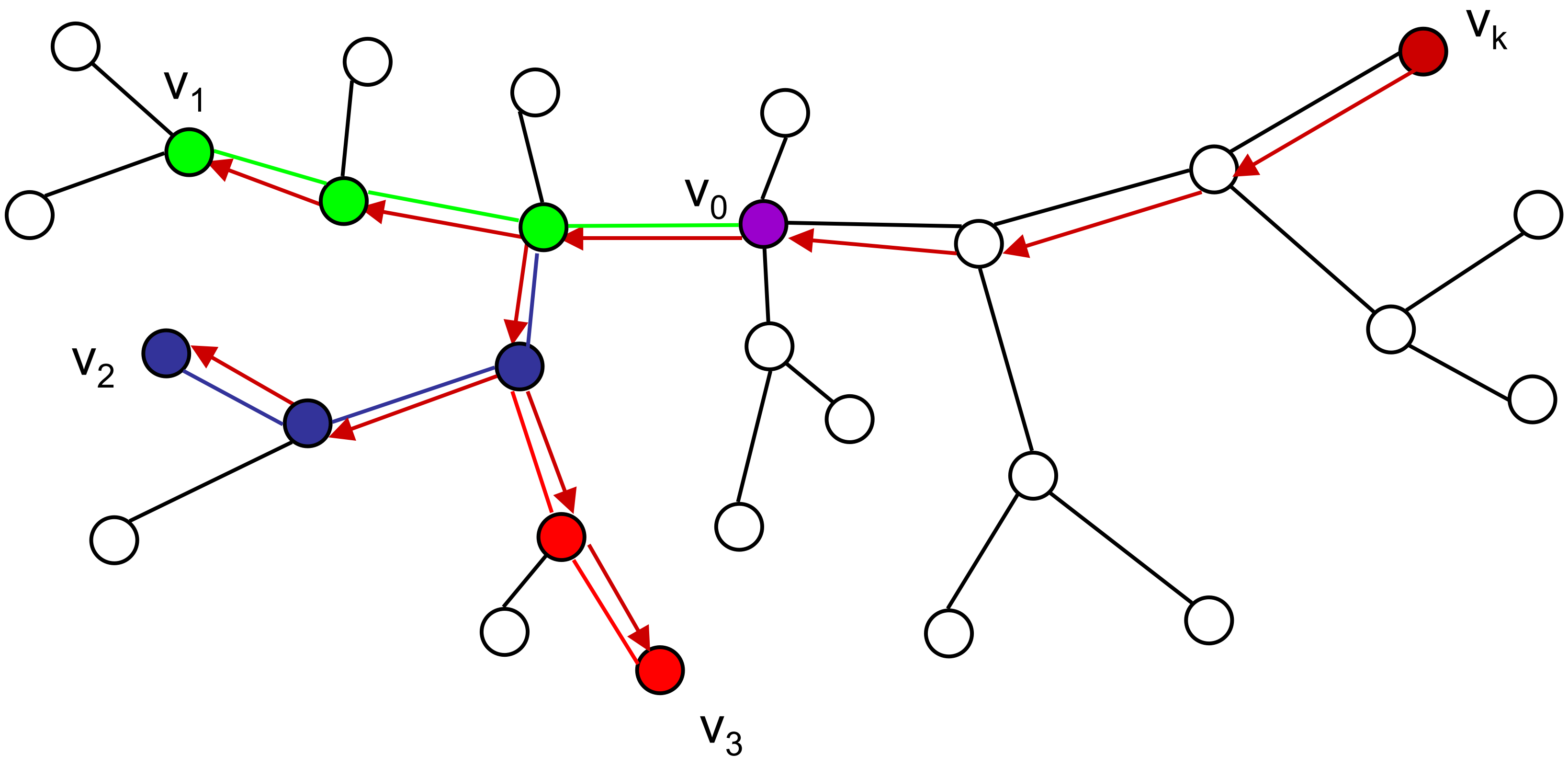
Example and Analysis

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



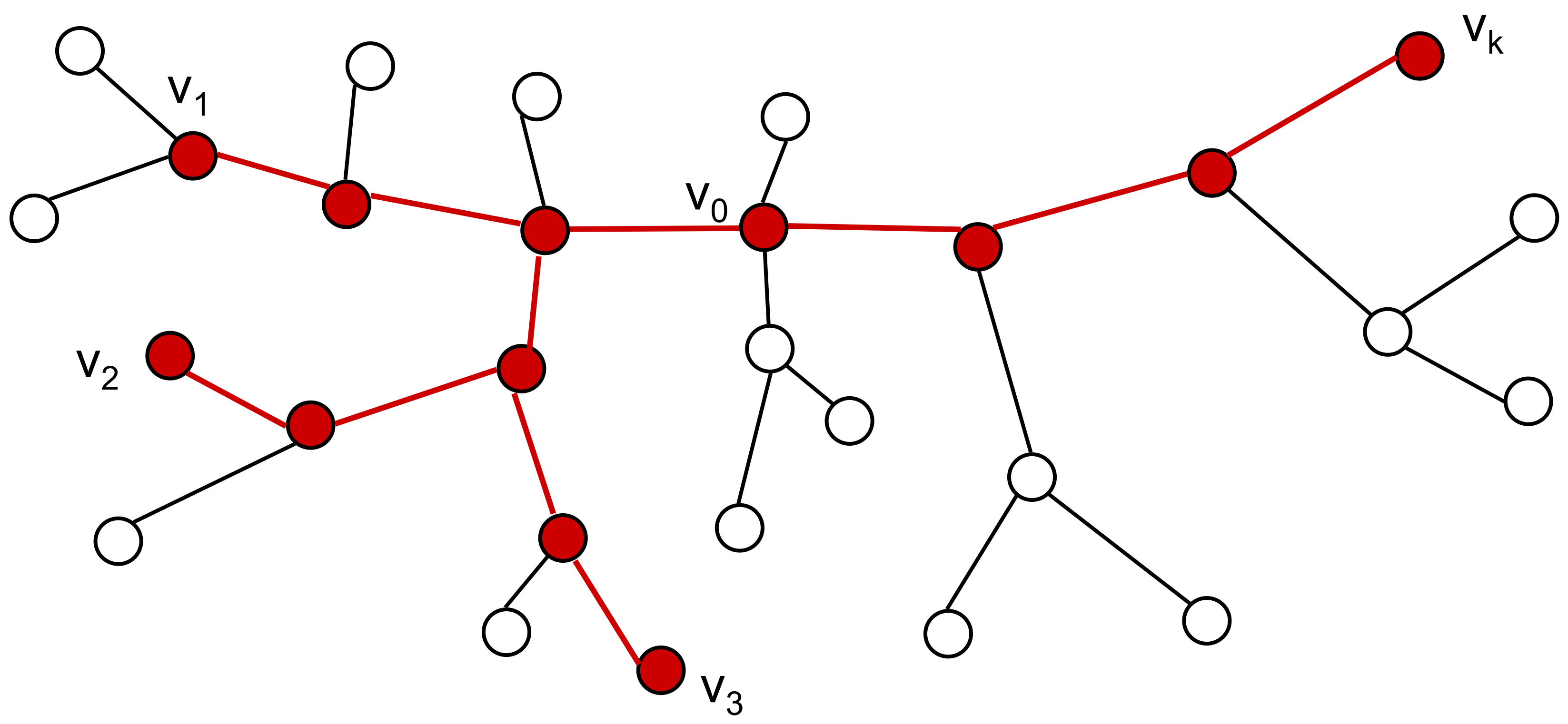
Example and Analysis

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



Scheme is 3-competitive (for a fixed tree)

Consider phase write (v_0), read (v_1), read (v_2), ... , read (v_{k-1}), write (v_k)



- Each strategy has to use each link of **the red subtree** at least *once*.
- Our strategy uses each of these links at most *three times*.