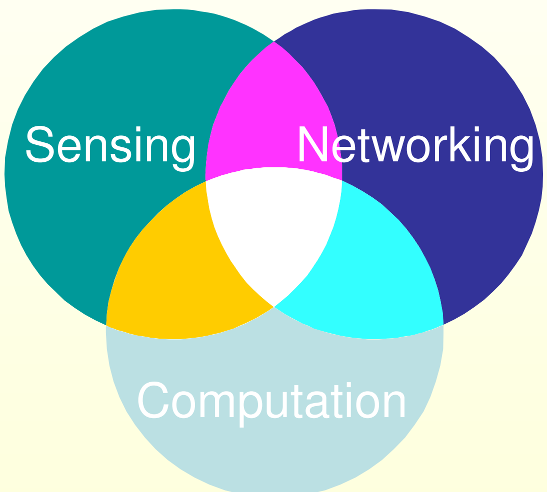
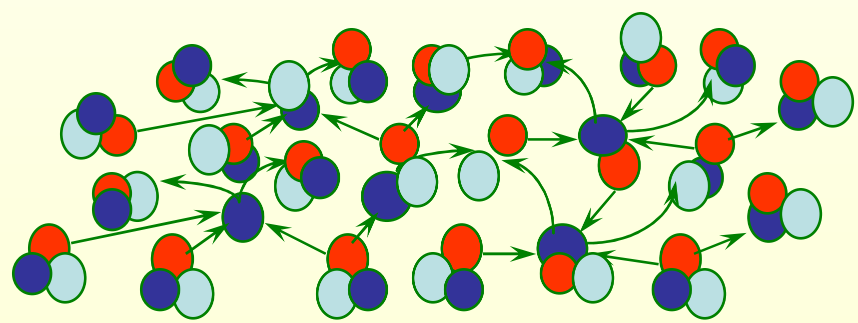
Information Brokerage



Leonidas Guibas
Stanford University



CS428

Next Class

- Thursday, April 28, 3:15 – 5:05 pm, in Gates 104

Project Schedule

- E-mail with team members (groups of up to three students), one short paragraph description by Fri., April 22
- PDF with detailed project description (3-4 pages) by Wed., May 4
- PDF with final write up by Wed., May 25
- Final project demos May 26 - June 2

Information Brokerage Services in Dynamic Environments

Information Brokerage

- Information providers (sources, producers) and information seekers (sinks, consumers) need ways to find out about and rendez-vous with each other
- Example: Surveillance from a remote node *r*:
  - e.g.: ***Send to r reports about animal detections in region A every t seconds***
    - Interrogation is propagated to sensor nodes in region *A*
    - Sensor nodes in region *A* are tasked to collect data
    - Data is sent back to the requestor *r* every *t* seconds

The Challenge is a Dynamic Environment [From D. Estrin]

- The physical world is highly dynamic
  - Dynamic operating conditions (sensing, networking)
  - Dynamic availability of resources
    - … *particularly energy*!
  - Dynamic tasks
- Devices must adapt automatically to the current environment and the task requirements
  - Too many devices for manual configuration
  - Environmental conditions are unpredictable
- Unattended and untethered operation is key for many applications

Data-Centric Paradigm

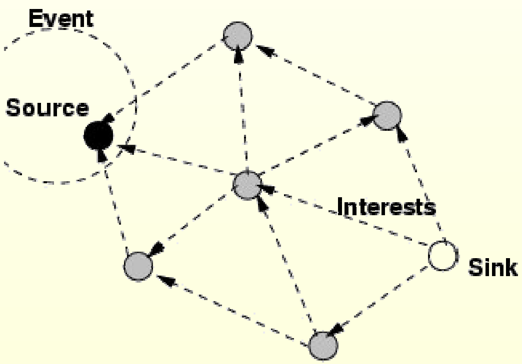
- Data Centric
  - The sensor network is queried for *specific data*
  - No *sensor-specific* query
  - Identity of source of data irrelevant
- Application Specific
  - In-sensor *processing*
  - In-sensor *caching*

- Localized Algorithms
  - Achieve global objective through local coordination

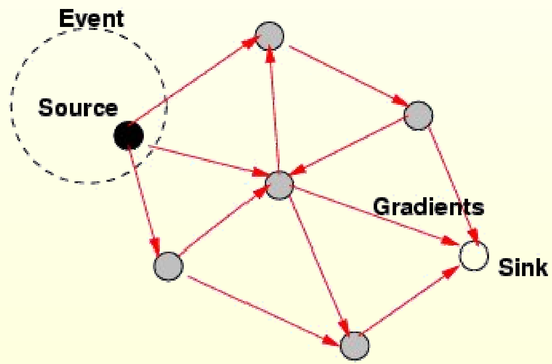
Approach

- Energy is the bottleneck resource
  - And communication is a major consumer – need to avoid communication over long distances
- Pre-configuration based on detailed global knowledge is rarely applicable
  - Achieve desired global behavior through localized interactions
  - Must empirically adapt to observed environment
- Leverage points
  - Small-form-factor nodes, densely distributed to achieve physical proximity to sensed phenomena
  - Application-specific, data-centric networks
  - Data processing/aggregation inside the network

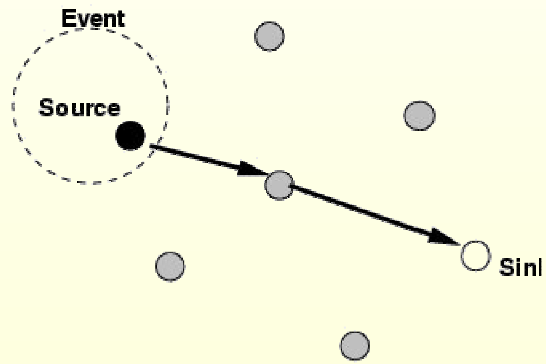
Directed Diffusion
[Intanagonwiwat, Govindan, Estrin '00]



(a) Interest propagation



(b) Initial gradients set up



(c) Data delivery along reinforced path

Directed Diffusion Concepts

- Application-aware communication primitives
  - expressed in terms of named data *(not in terms of the nodes generating or requesting data)*
- A consumer of data, a sink node, initiates an interest in data with certain attributes
- Nodes diffuse the interest towards data producers (sources), via a sequence of local interactions
- This process sets up gradients in the network which channel the delivery of data
- Reinforcement (positive and negative) is used to converge to efficient routes
- Intermediate nodes opportunistically fuse interests, aggregate, correlate, or cache data ...

Data Naming

- Content-based naming
  - Data is named by attribute–value pairs
  - This makes matching of interests with data simple
  - Selecting a naming scheme important and more complex ones can be considered
  - The nodes where information resides are not part of the naming scheme

Request: Interest

```
type = four-legged animal
   interval = 20 ms
   duration = 10 seconds
   rect = [-100,100,200,200]
```

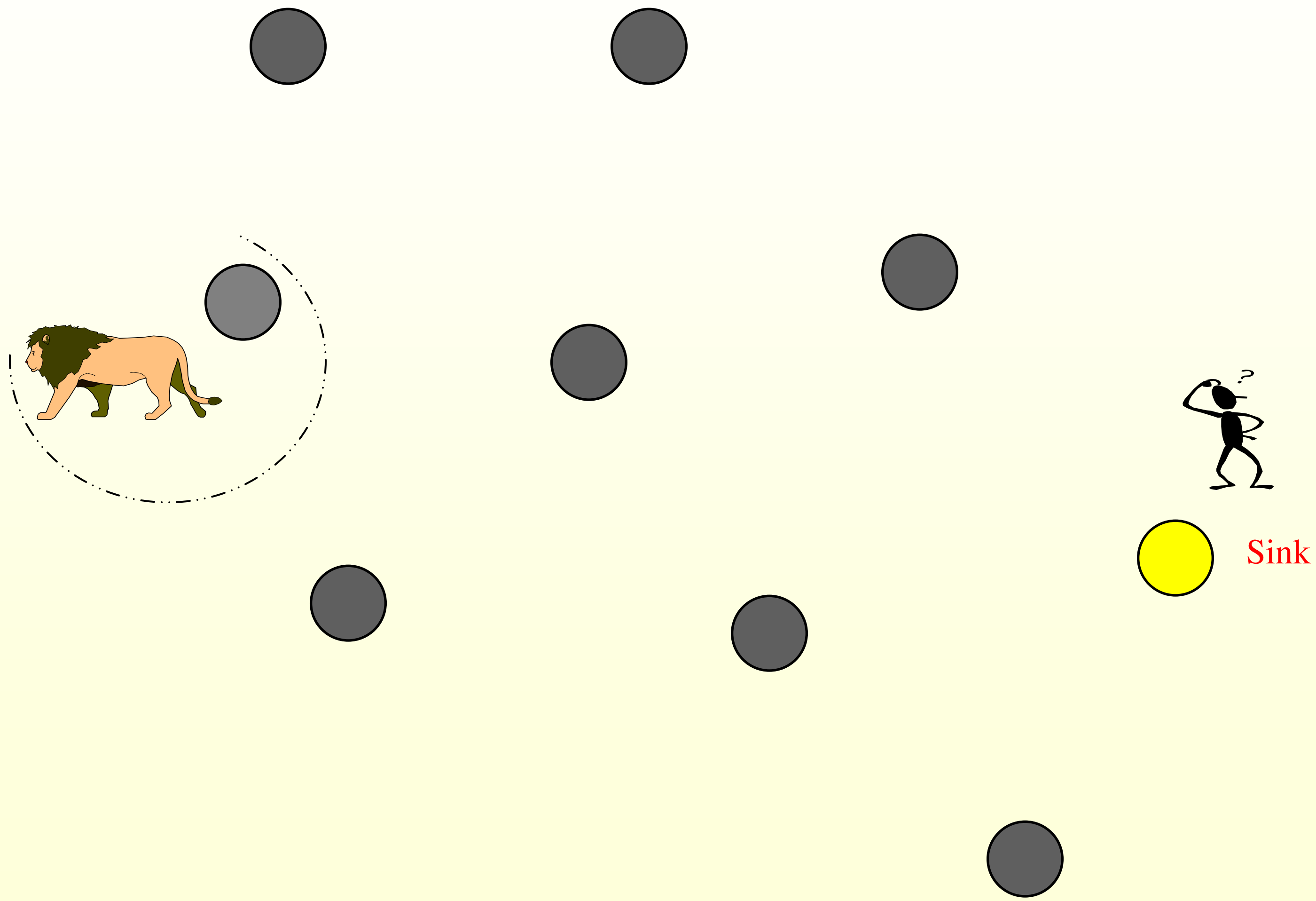
Reply: Data

```
type = four-legged animal
   instance = elephant
   location = [125, 220]
        Intensity = 0.6
        confidence = 0.85
        timestamp = 01:20:40
```

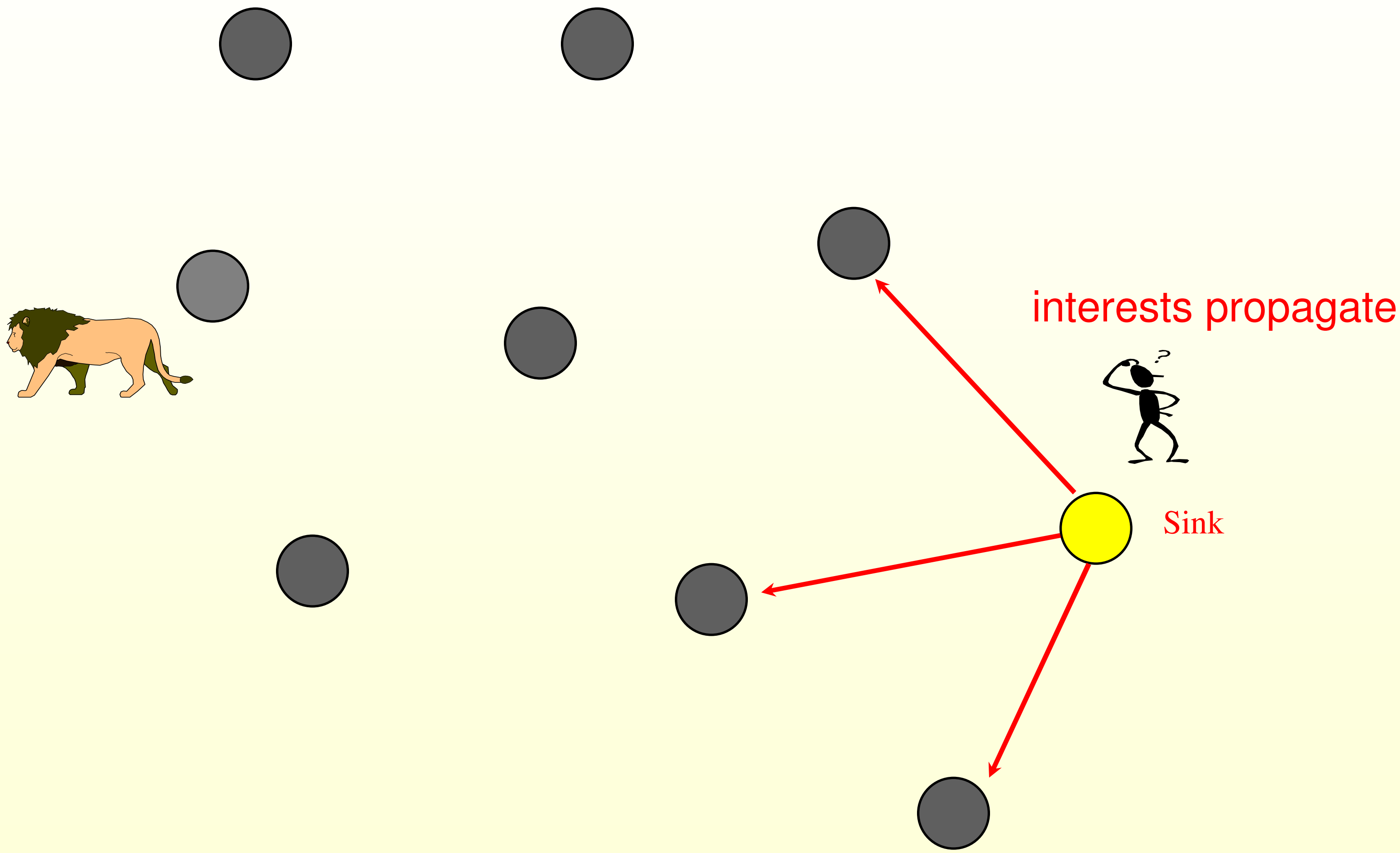
Interests and Gradients

- Interests describe data needed by a node in the sensor network
  - Interests are injected into the network at sinks.
  - Sinks broadcast the interest.
  - An interval specifies the event data rate desired.
  - *Initially, requested intervals are much larger than needed.*
  - Each node maintains an interest cache.
- Each cache interest entry also contains gradients.
  - Specifies a data rate and a direction of data flow for each requesting neighbor
  - Data flows from the source to the sink along the gradient links

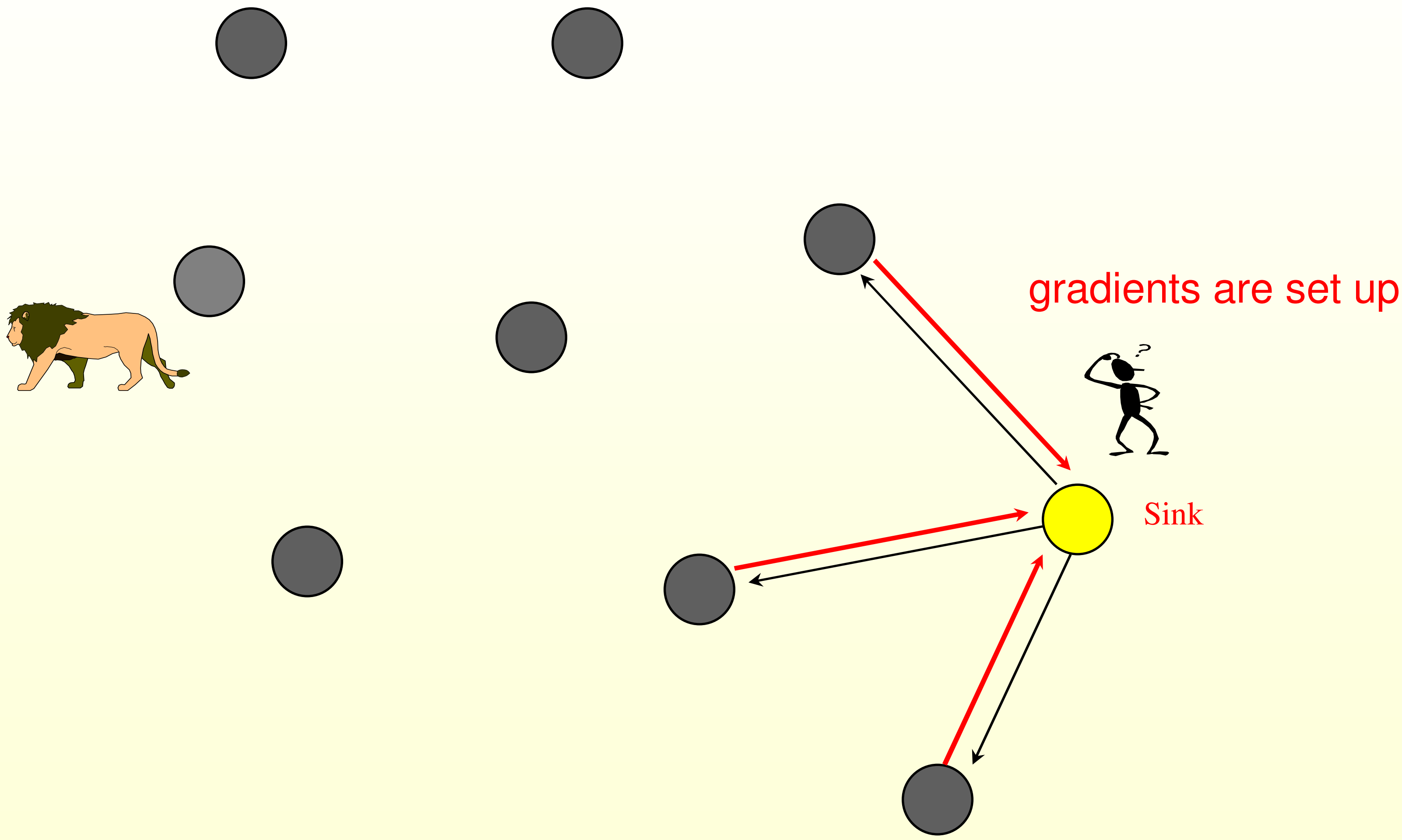
Illustrating Directed Diffusion



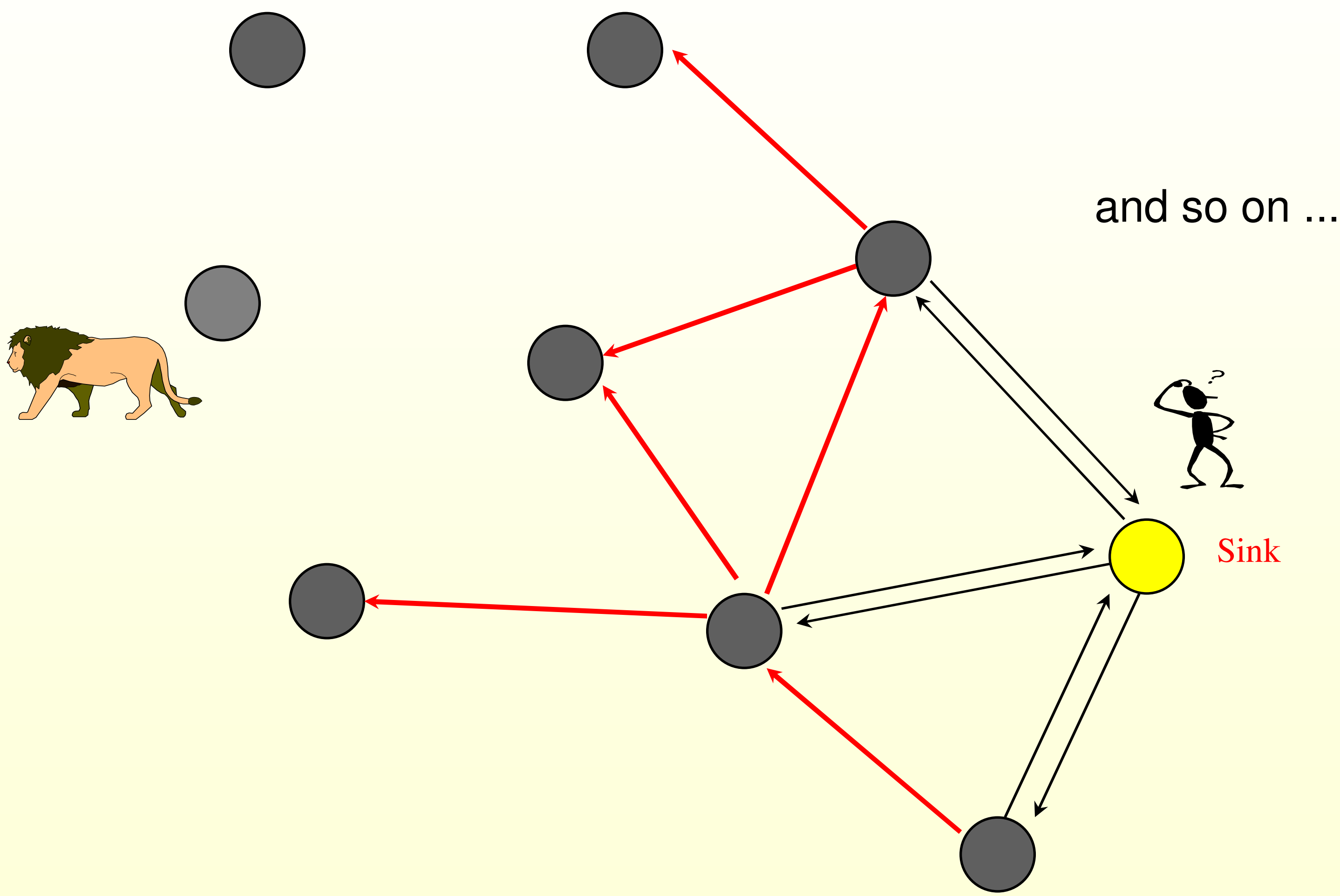
Illustrating Directed Diffusion



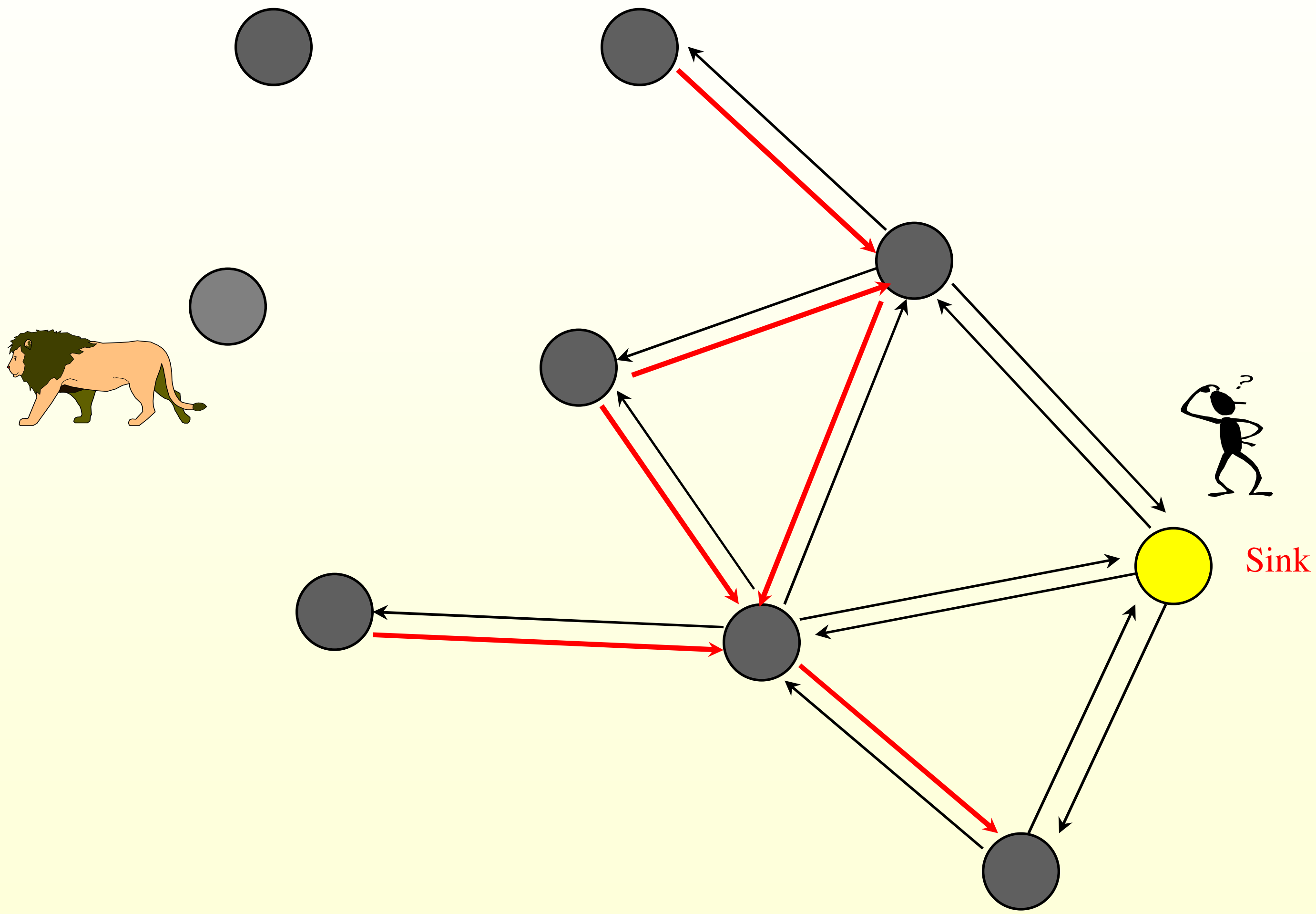
Illustrating Directed Diffusion



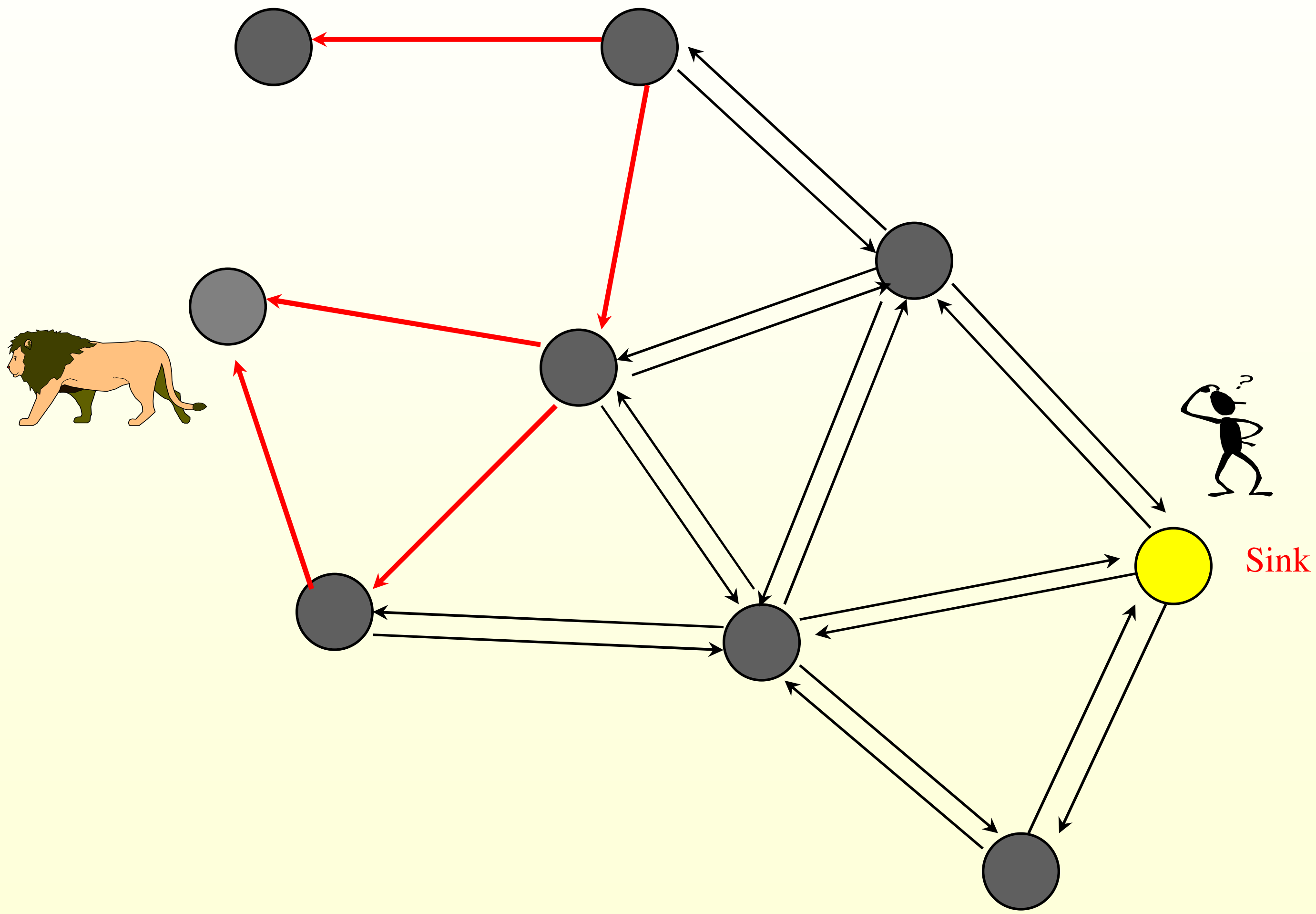
Illustrating Directed Diffusion



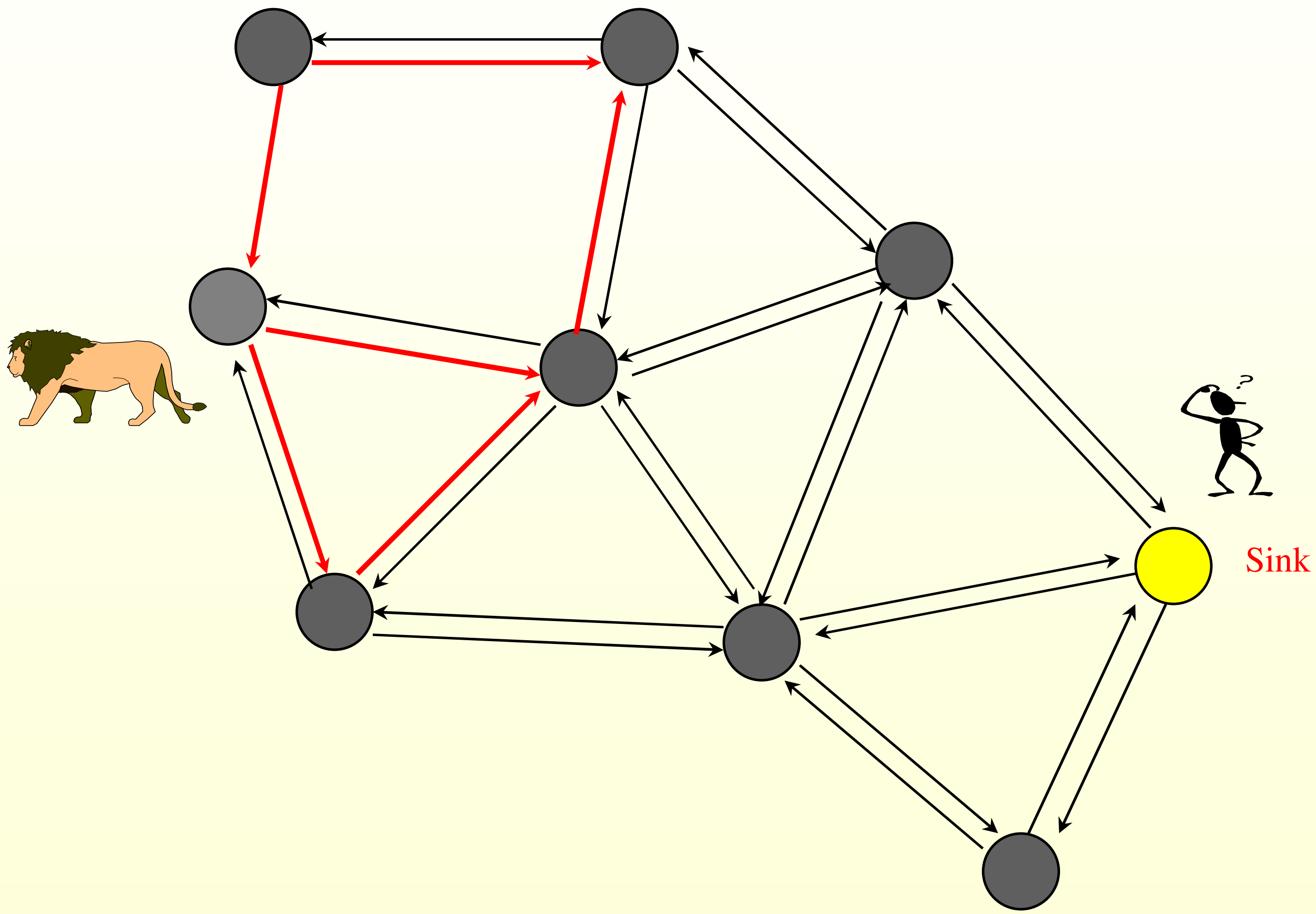
Illustrating Directed Diffusion



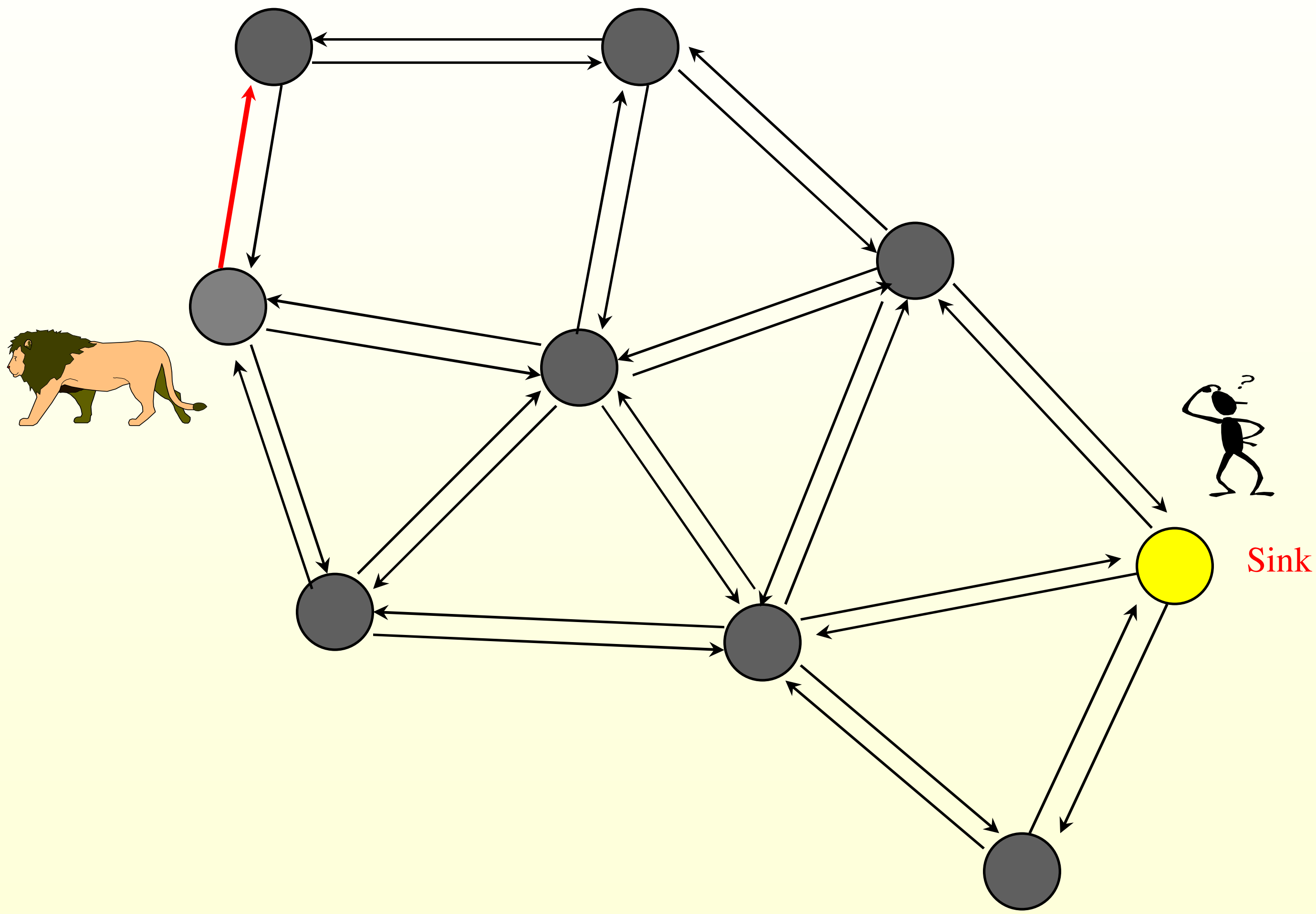
Illustrating Directed Diffusion



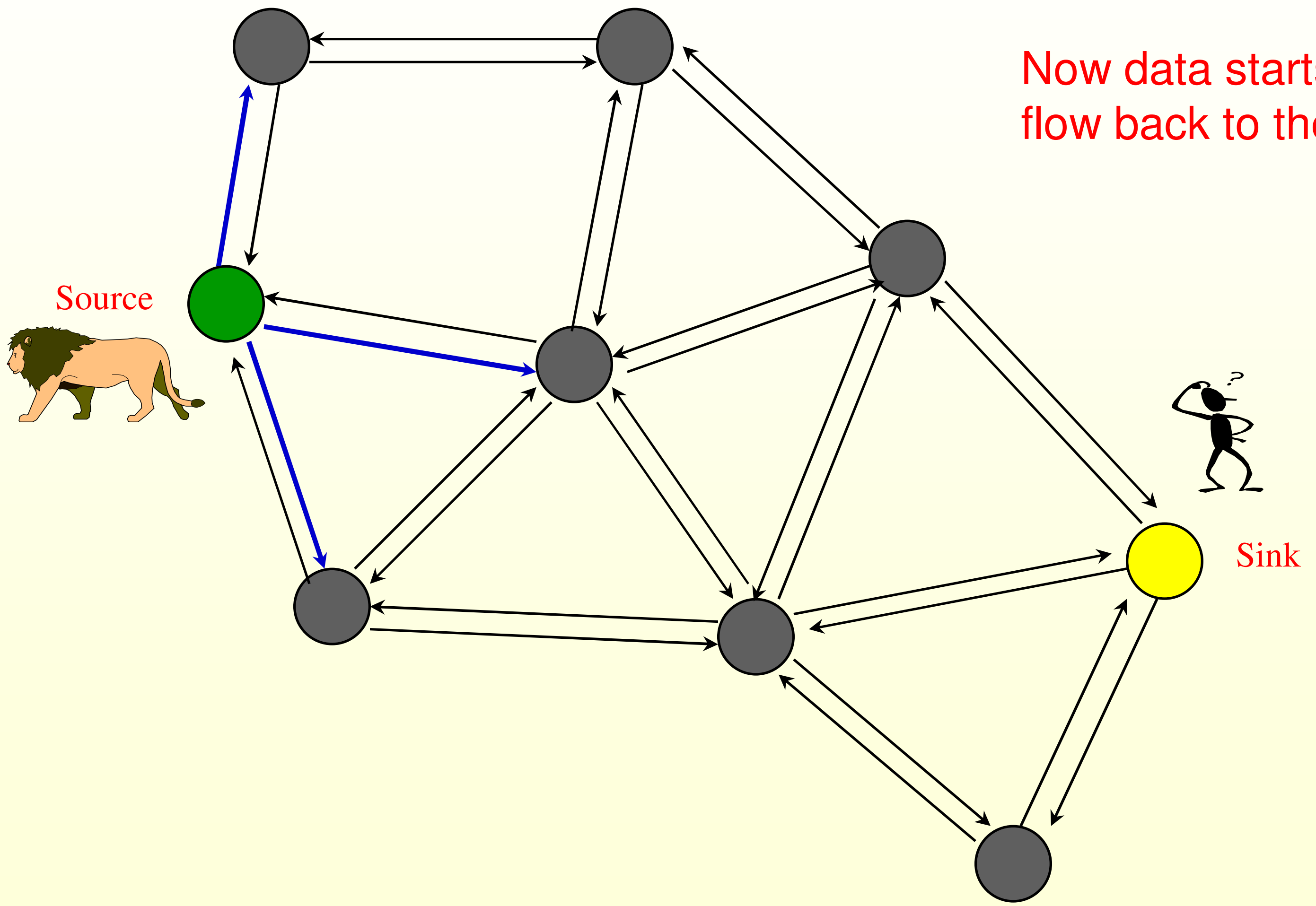
Illustrating Directed Diffusion



Illustrating Directed Diffusion

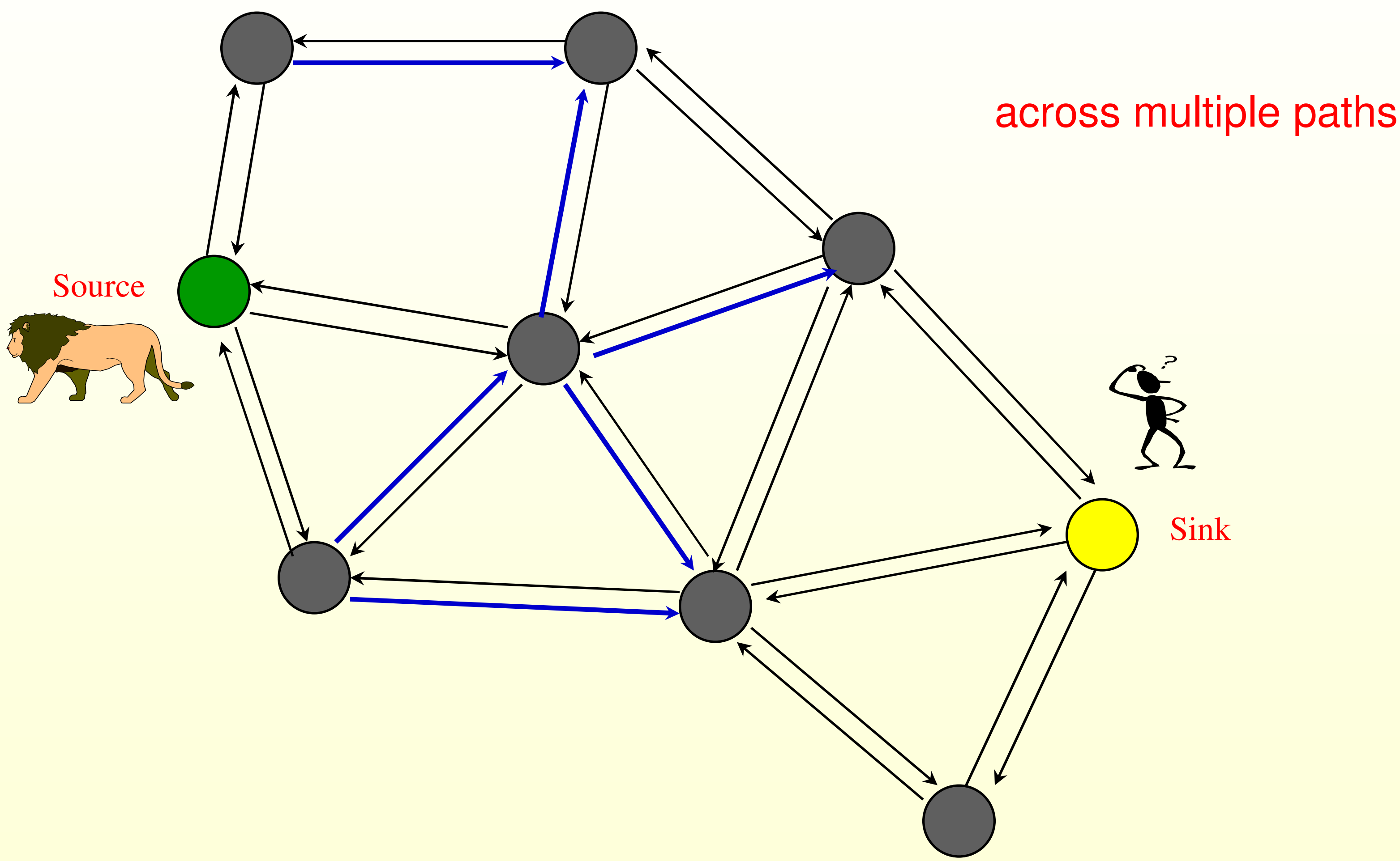


Illustrating Directed Diffusion

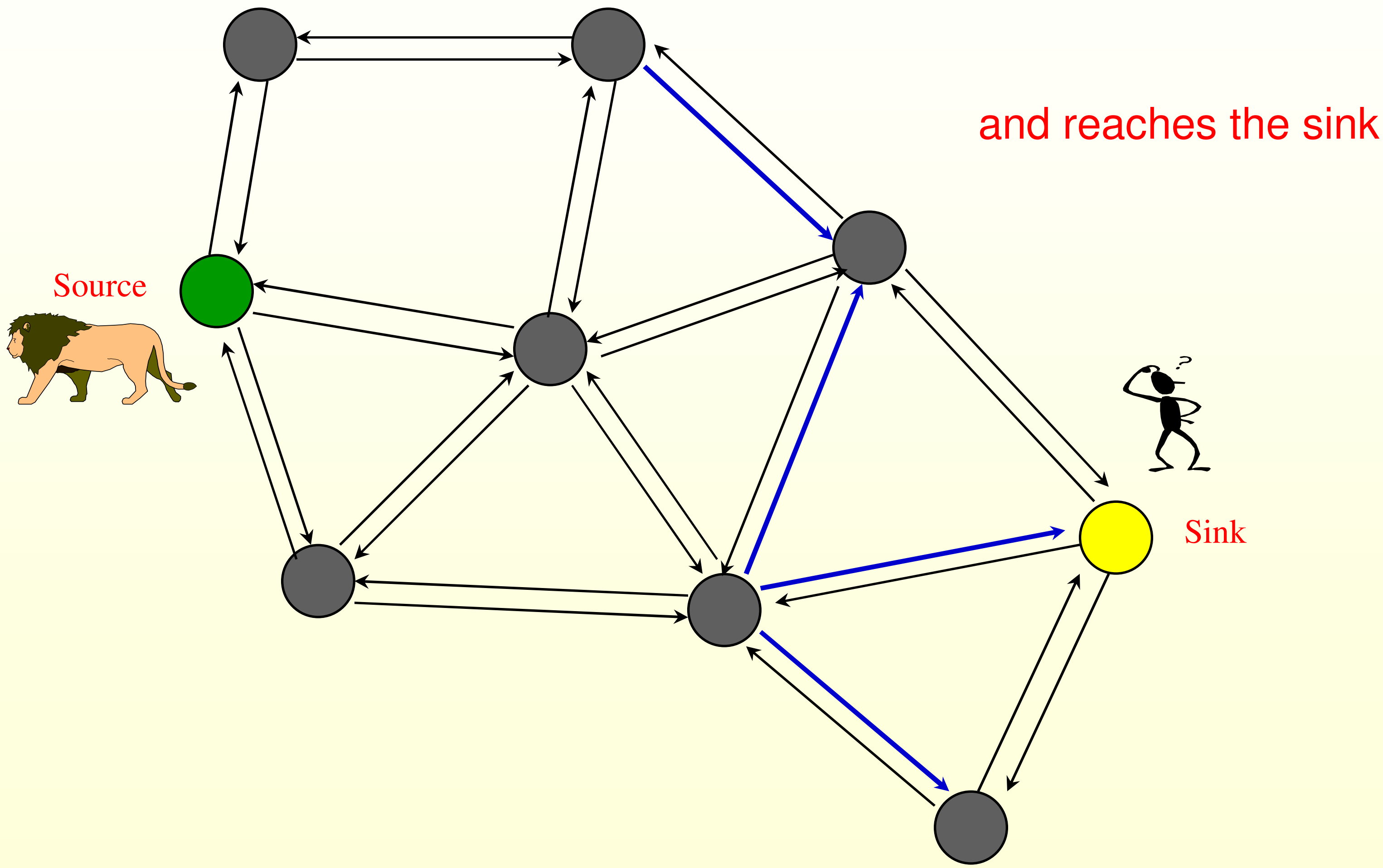


Now data starts to flow back to the sink

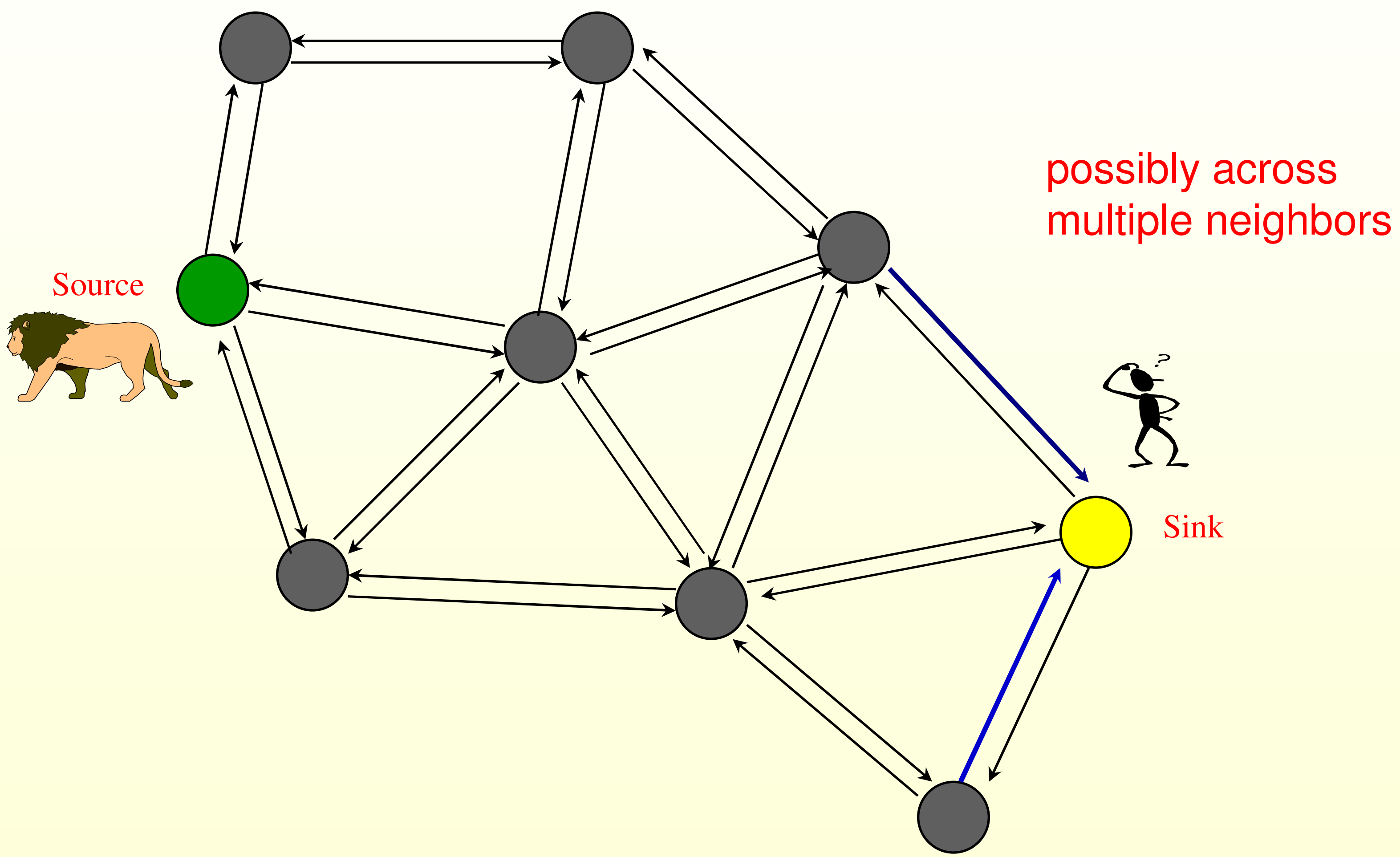
Illustrating Directed Diffusion



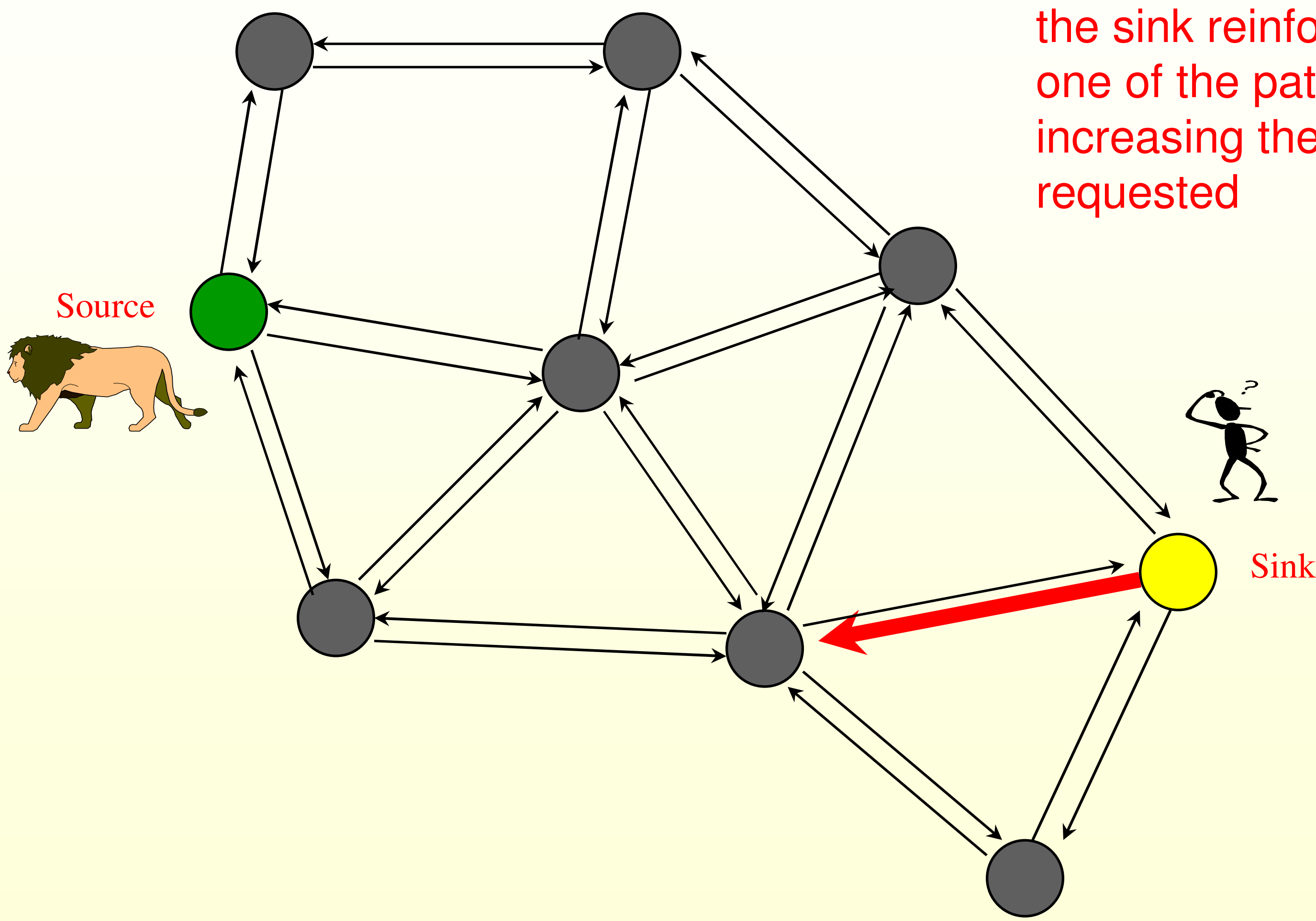
Illustrating Directed Diffusion



Illustrating Directed Diffusion

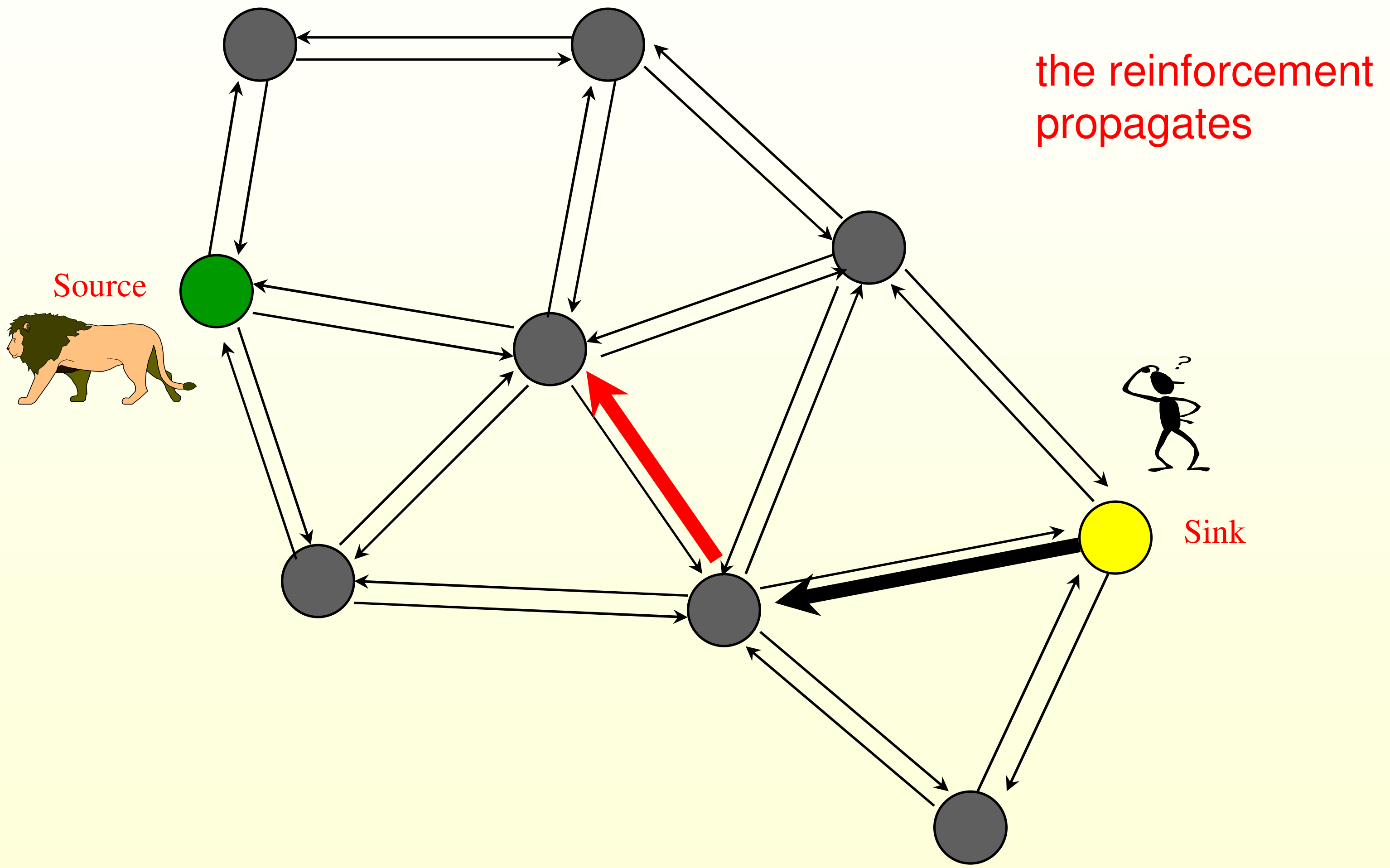


Illustrating Directed Diffusion

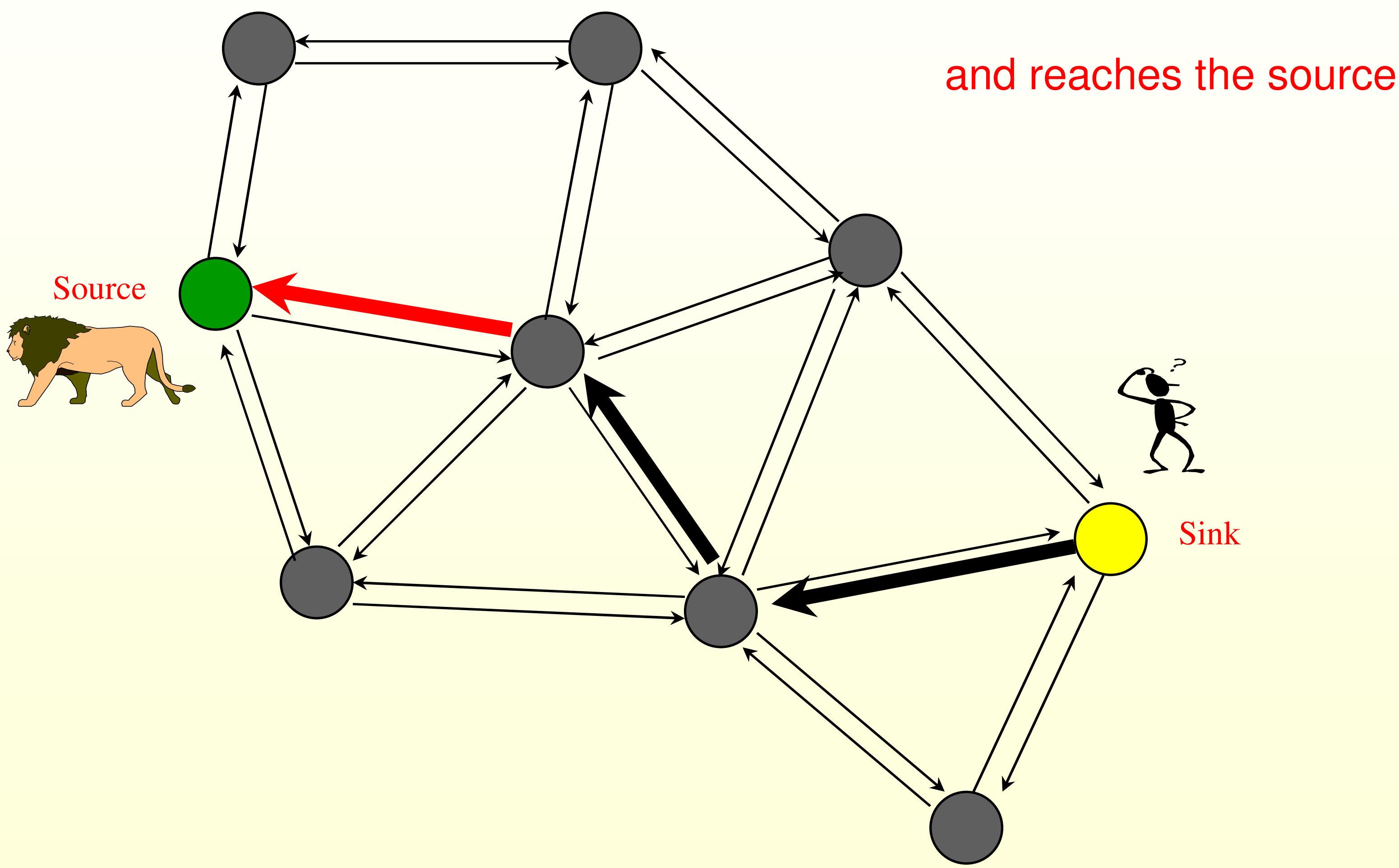


the sink reinforces one of the paths, by increasing the data rate requested

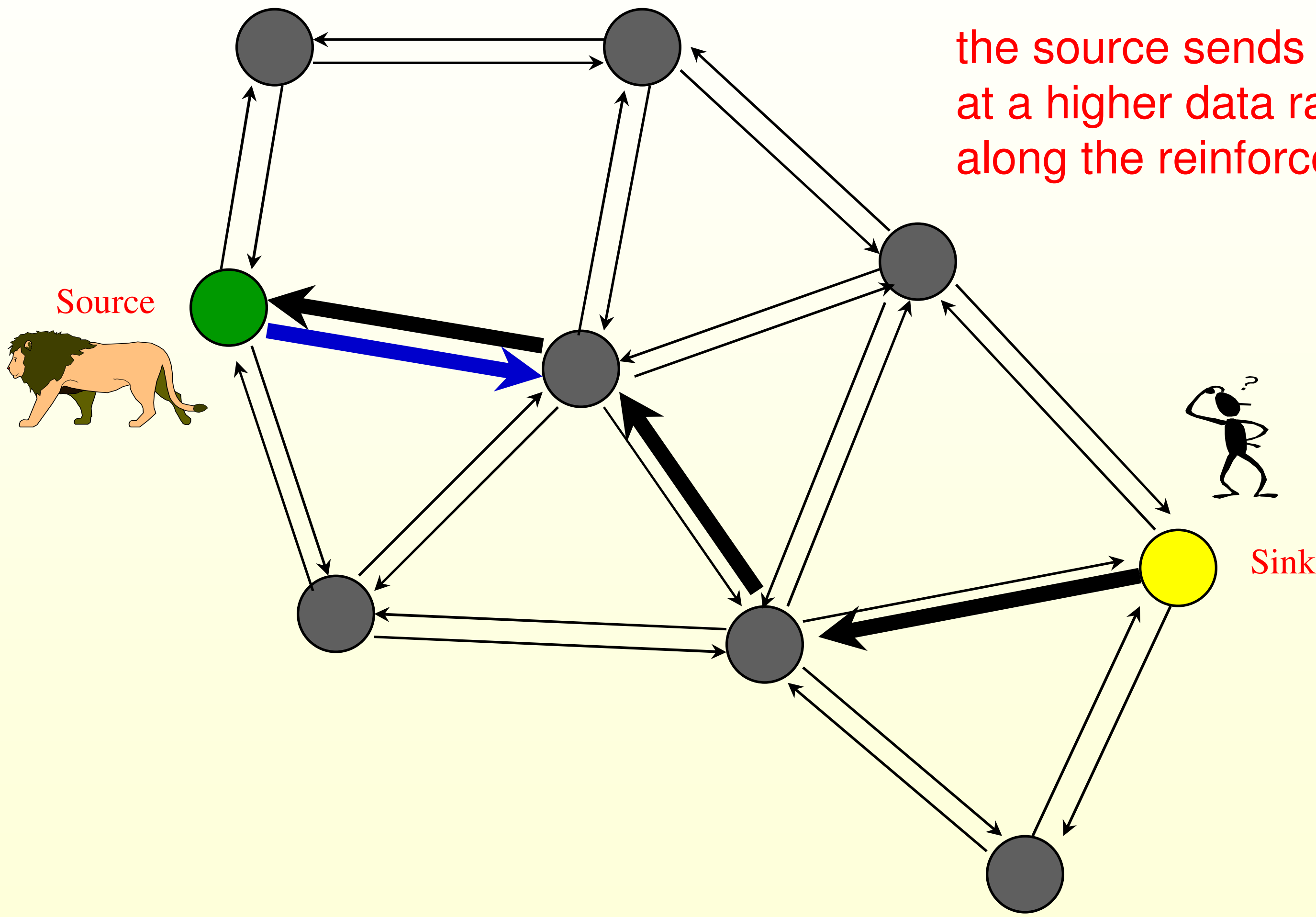
Illustrating Directed Diffusion



Illustrating Directed Diffusion

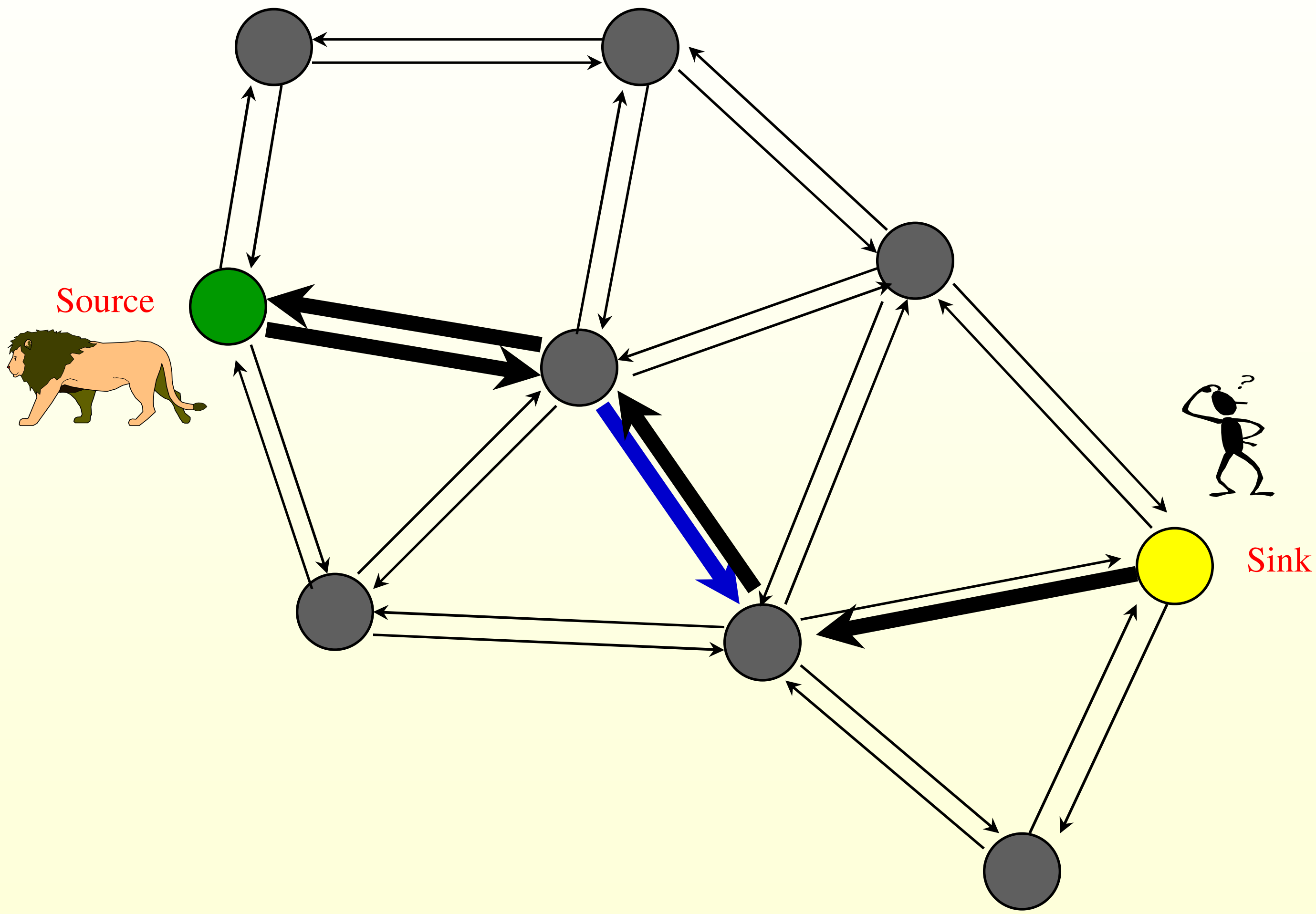


Illustrating Directed Diffusion

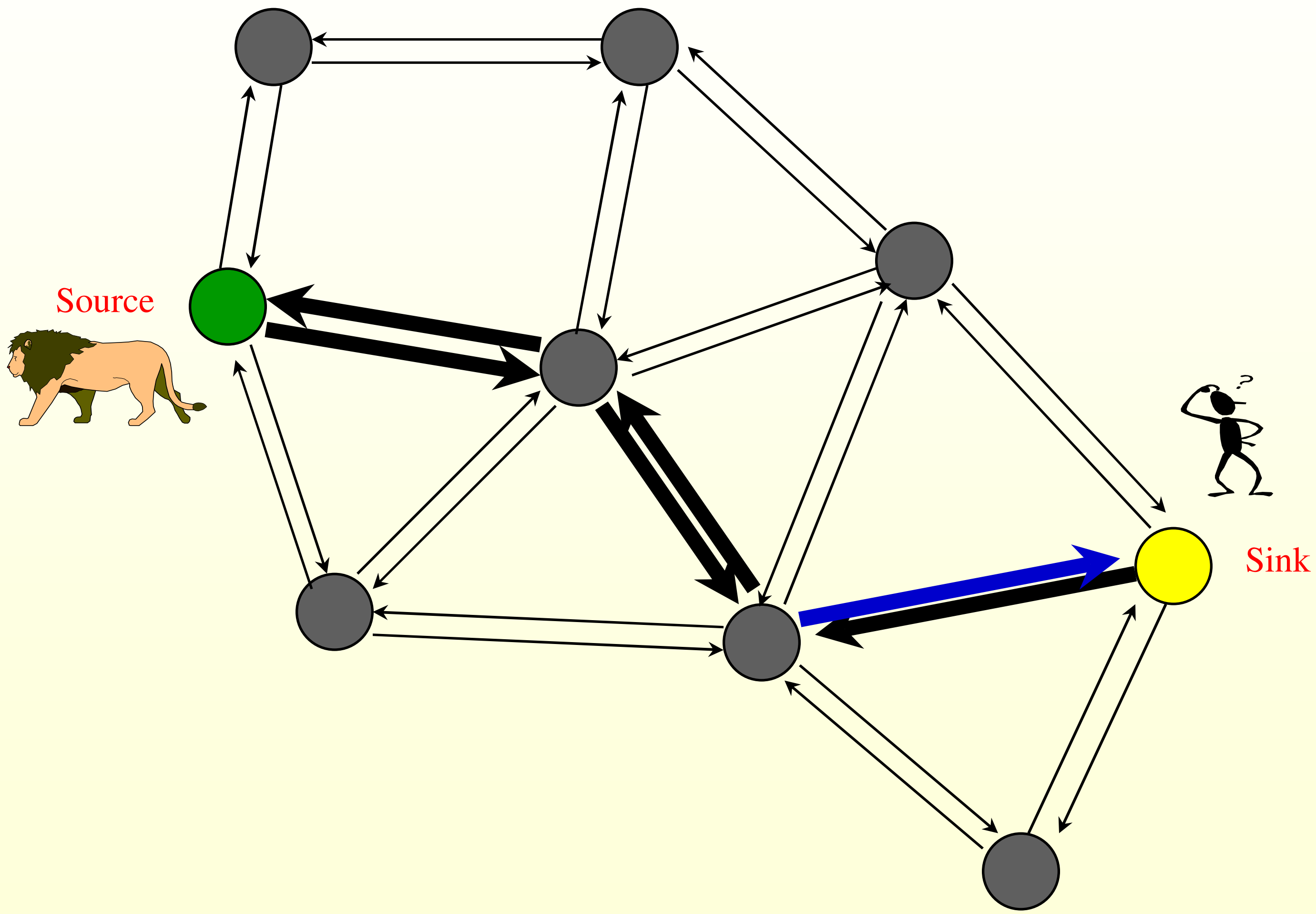


the source sends data
at a higher data rate
along the reinforced path

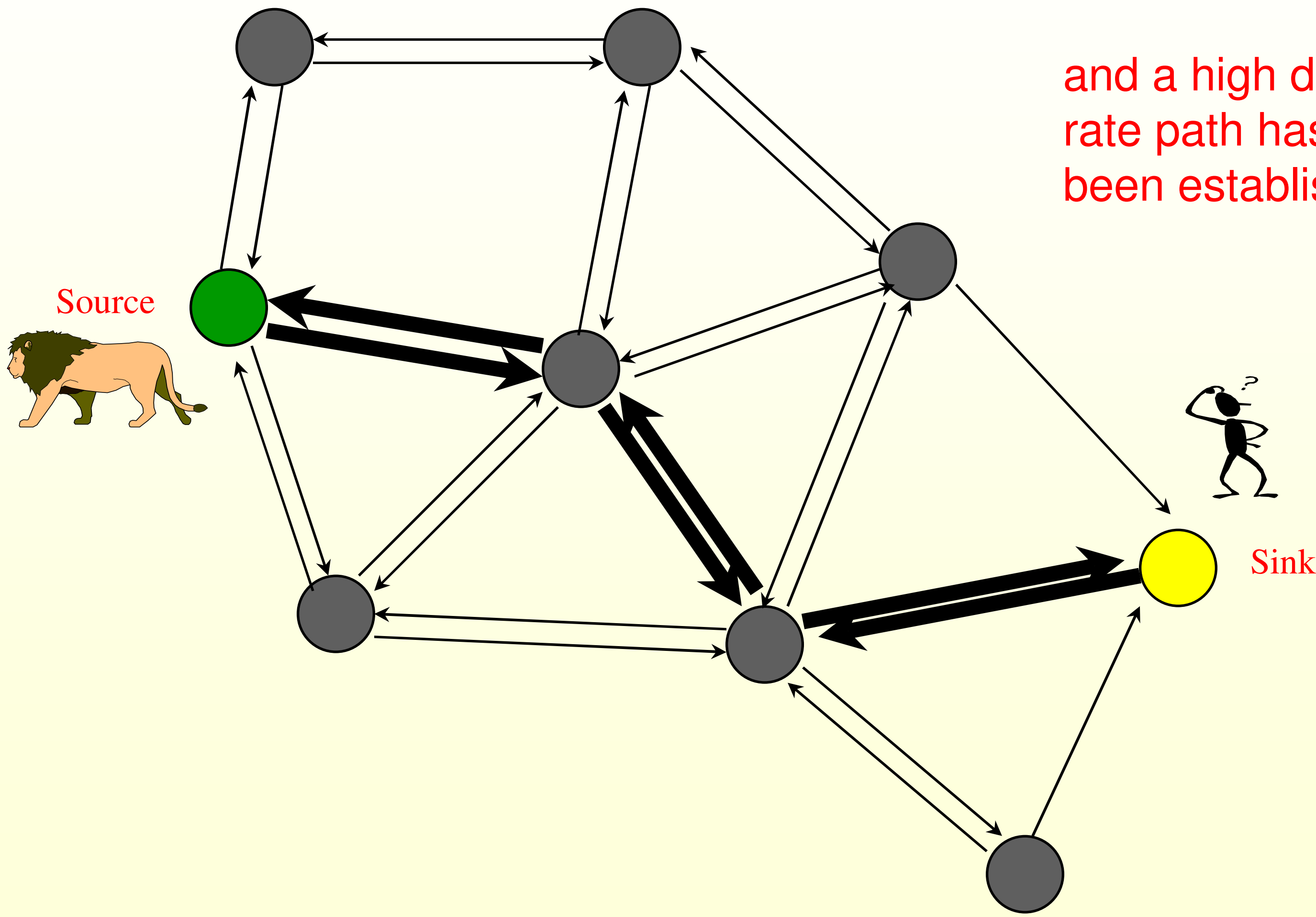
Illustrating Directed Diffusion



Illustrating Directed Diffusion

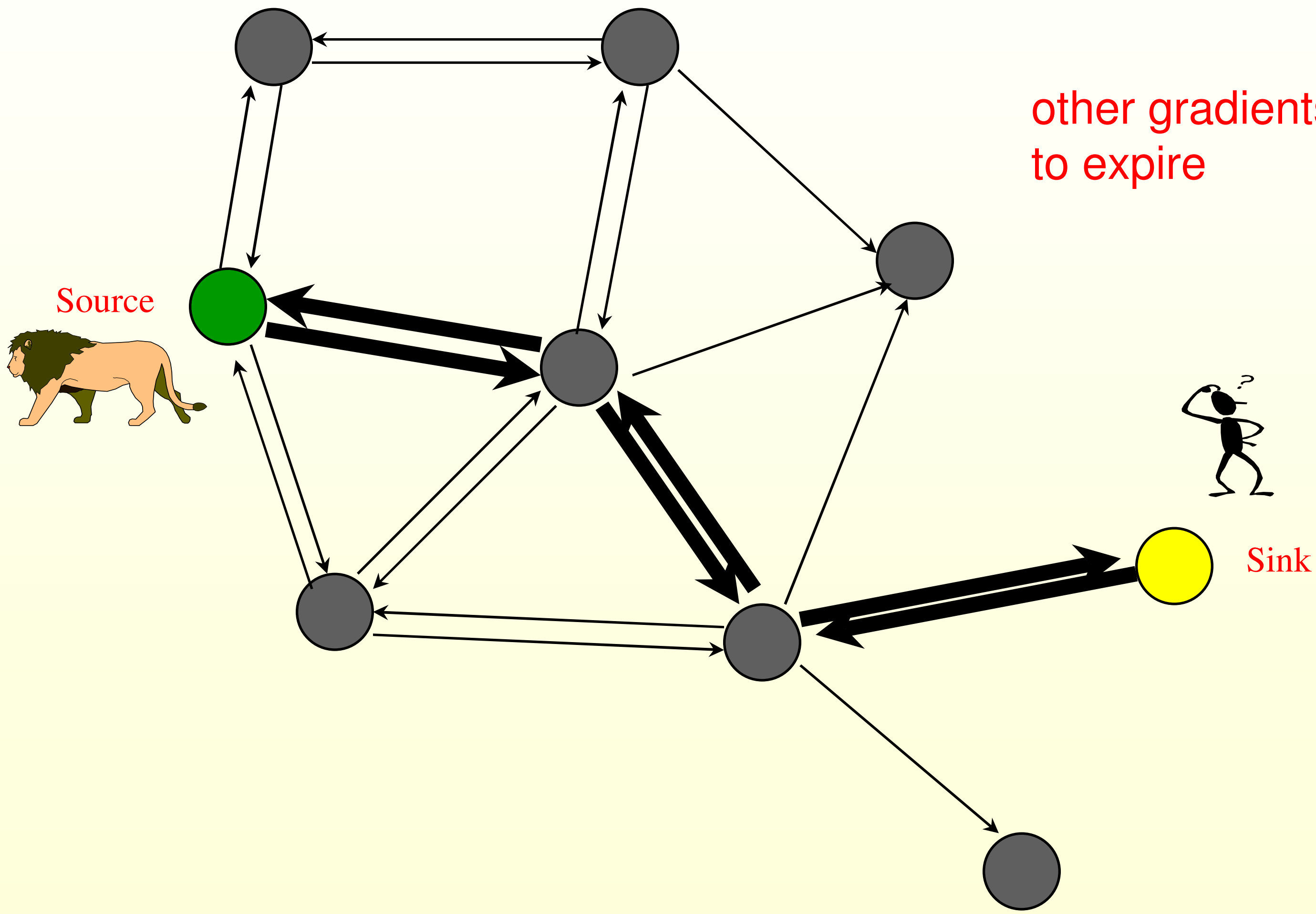


Illustrating Directed Diffusion



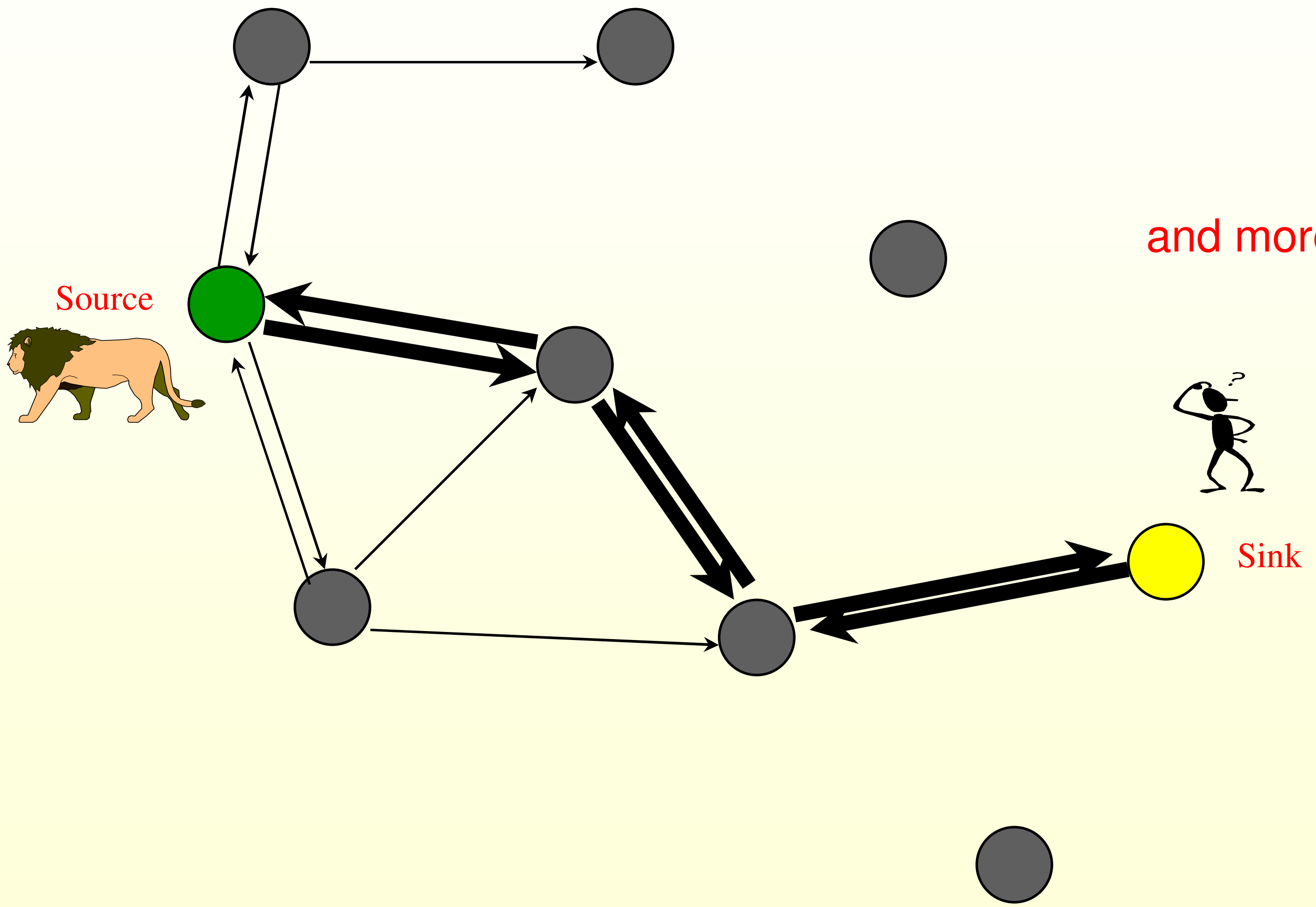
and a high data rate path has been established

Illustrating Directed Diffusion



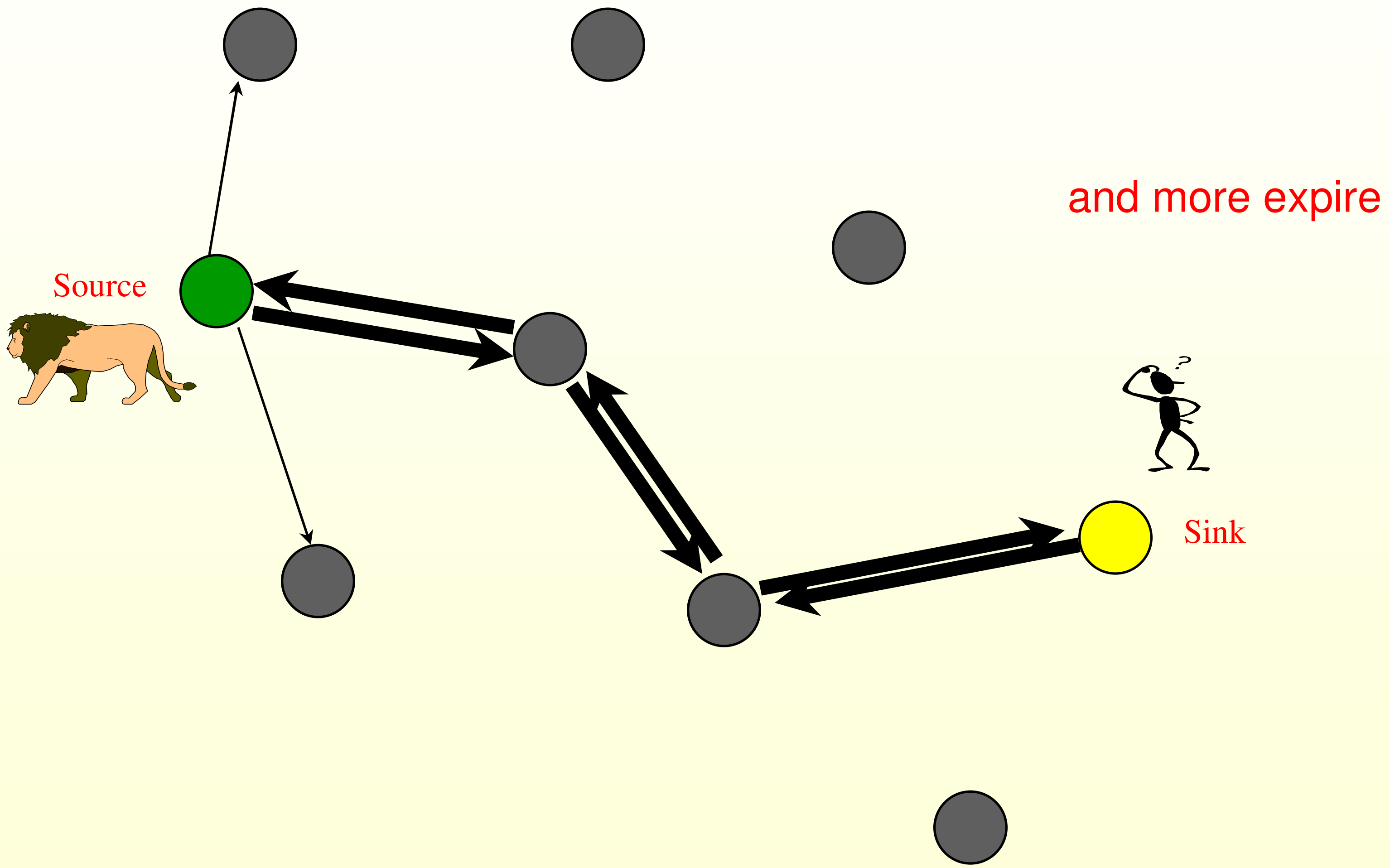
other gradients start to expire

Illustrating Directed Diffusion

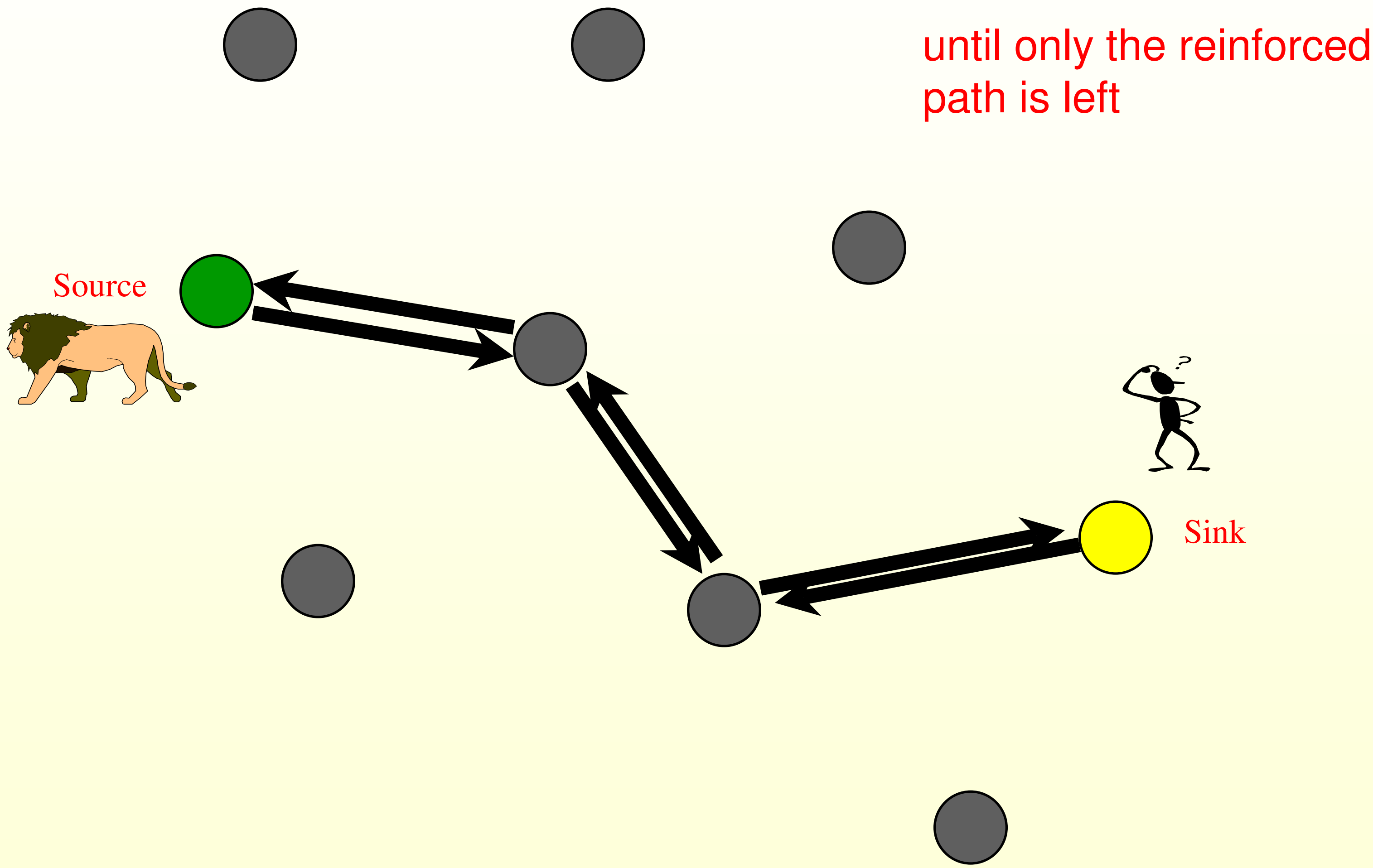


and more expire

Illustrating Directed Diffusion

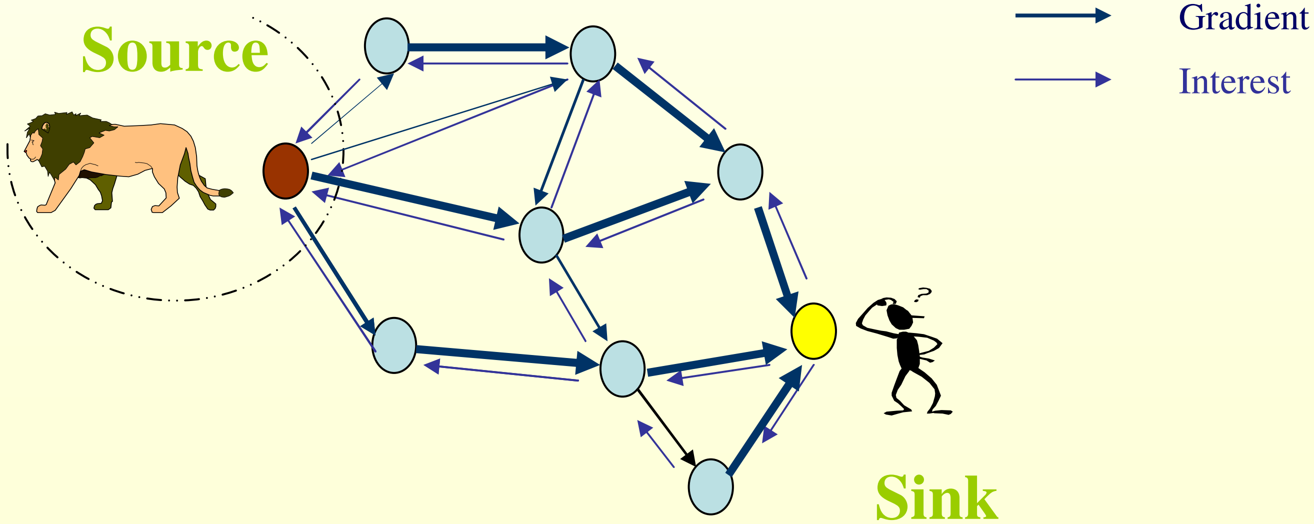


Illustrating Directed Diffusion



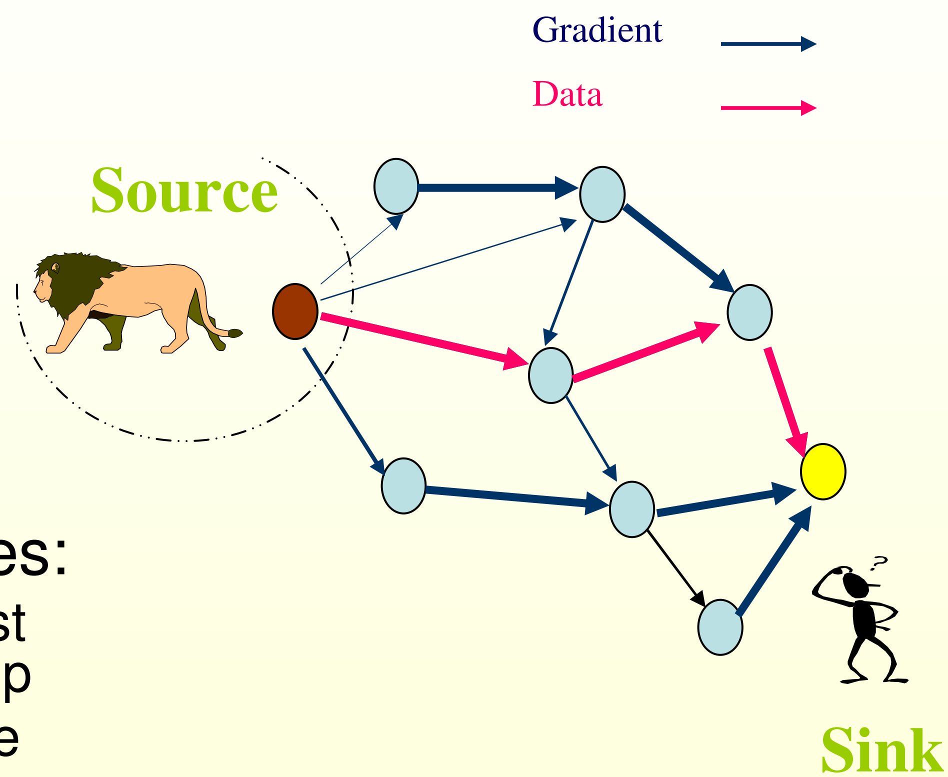
Interest Propagation

- Involves flooding the network
- Could use constrained or bidirectional flooding based on source location.
- Directional propagation can also be based on previously cached data.



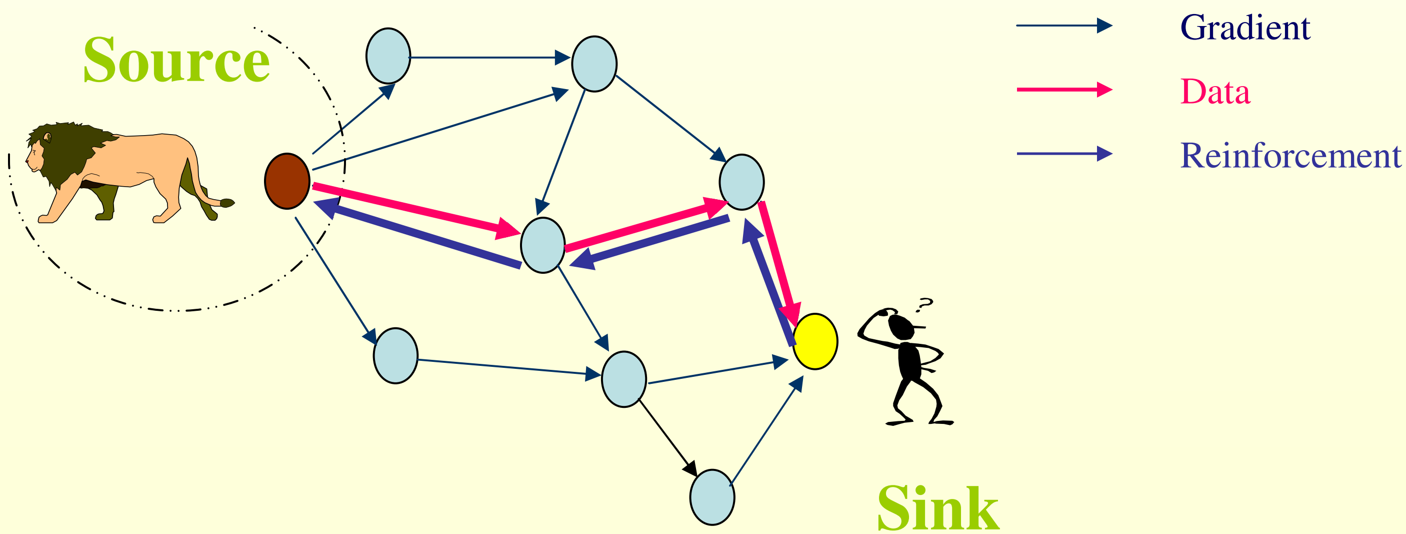
Data Propagation

- Source nodes match signature waveforms from codebook against observations
- Nodes match data against interest cache, compute highest event-rate request from all gradients, and (re)-sample events at this rate
- Intermediate receiving nodes:
  - Find matching entry in interest cache; no match → silent drop
  - Check and update data cache (loop prevention, aggregation, duplicate suppression, etc.)
  - Retrieve all gradients, and resend message, performing frequency conversion if necessary



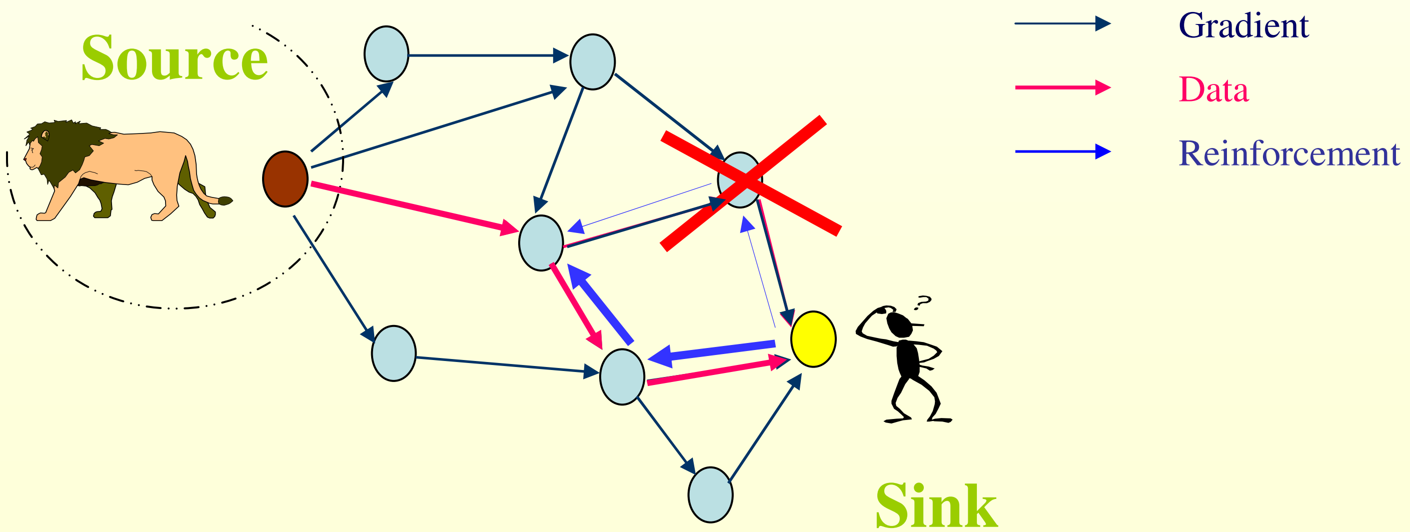
Reinforcement

- Reinforce one of the neighbor after receiving initial data.
  - Neighbor(s) from whom new events are received.
  - Neighbor who is consistently performing better than others.
  - Neighbor from whom most events are received.

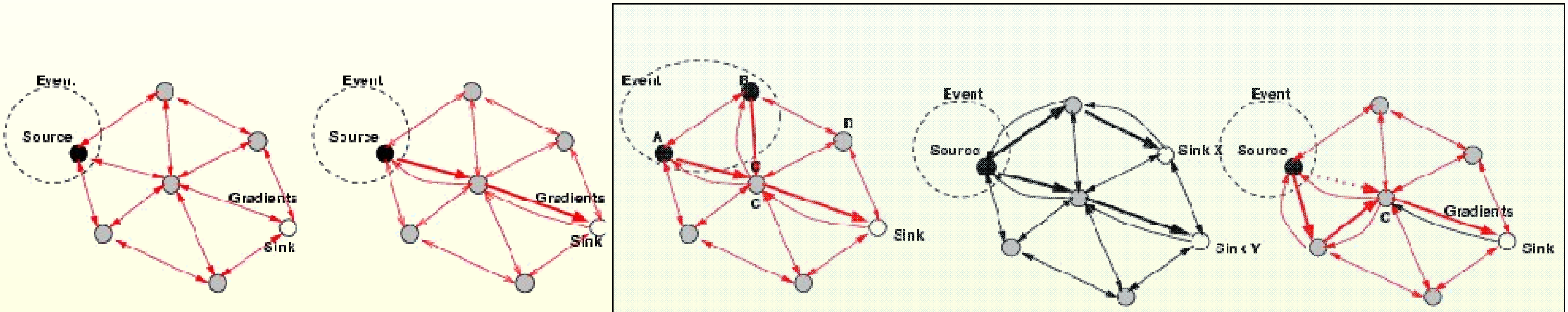


Negative Reinforcement

- Explicitly degrade some paths by re-sending *interests* with lower data request rates.
- Cache entries time out if not reinforced



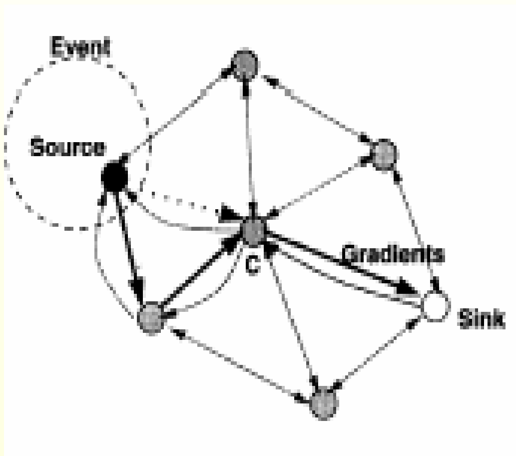
Other Aspects of Directed Diffusion



(a) Gradient establishment (b) Reinforcement (c) Multiple sources (d) Multiple sinks (e) Repair

Figure 2: Illustrating different aspects of diffusion.

Local Repair for Failed Paths



- Intermediate nodes on a previously reinforced path can apply reinforcement rules (useful for failed or degraded paths)
- C detects degradation
  - By noticing that the event reporting rate from its upstream neighbor (source) is now lower
  - By realizing that other neighbors have been transmitting previously unseen location estimates.
- And applies reinforcement rules
- Problem: wasted resources (e.g., if other downstream nodes also do the same)
- Avoid this by interpolating location estimates from the events

DD Scenario Notes

- Reinforcement (optimization):
  - Data-driven rules; ex., new msg. from neighbor → resend original with smaller sampling interval
  - This neighbor, in turn, reinforces upstream nodes
  - Local rule : minimize delay; other rules are possible
  - Passive negative reinforcement (timeouts) or active (negative weights)
- Multiple sources + reinforcement
  - Works in some cases, open for further exploration
- Multiple sinks: Exploit prior setup (i.e., use cache)
- Intermediate nodes use reinforcement for local repair
  - Cascading reinforcement discoveries from upstream can be a problem; one soln.: interpolate requests to preserve status-quo

Local Behavior Choices

1. For propagating interests
   - In our example, flood
   - More sophisticated behaviors possible: e.g. based on cached information, GPS
2. For setting up gradients
   - Highest gradient towards neighbor from whom we first heard interest
   - Others possible: towards neighbor with highest energy
3. For data transmission
   - Different local rules can result in single path delivery, striped multi-path delivery, single source to multiple sinks and so on.
4. For reinforcement
   - reinforce one path, or part thereof, based on observed losses, delay variances etc.
   - other variants: inhibit certain paths because resource levels are low

DD Design Space

Diffusion element	Design Choices
Interest Propagation	• Flooding • Constrained or directional flooding based on location • Directional propagation based on previously cached data
Data Propagation	• Reinforcement to single path delivery • Multipath delivery with selective quality along different paths • Multipath delivery with probabilistic forwarding
Data caching and aggregation	• For robust data delivery in the face of node failure • For coordinated sensing and data reduction • For directing interests
Reinforcement	• Rules for deciding when to reinforce • Rules for how many neighbors to reinforce • Negative reinforcement mechanisms and rules

Figure 3: Design Space for Diffusion

Initial Simulation Study of Diffusion

- Key metric
  - Average Dissipated Energy per event delivered
    - captures energy efficiency and network lifetime
- Compare directed diffusion to
  - flooding
  - centrally computed dissemination tree (omniscient multicast)

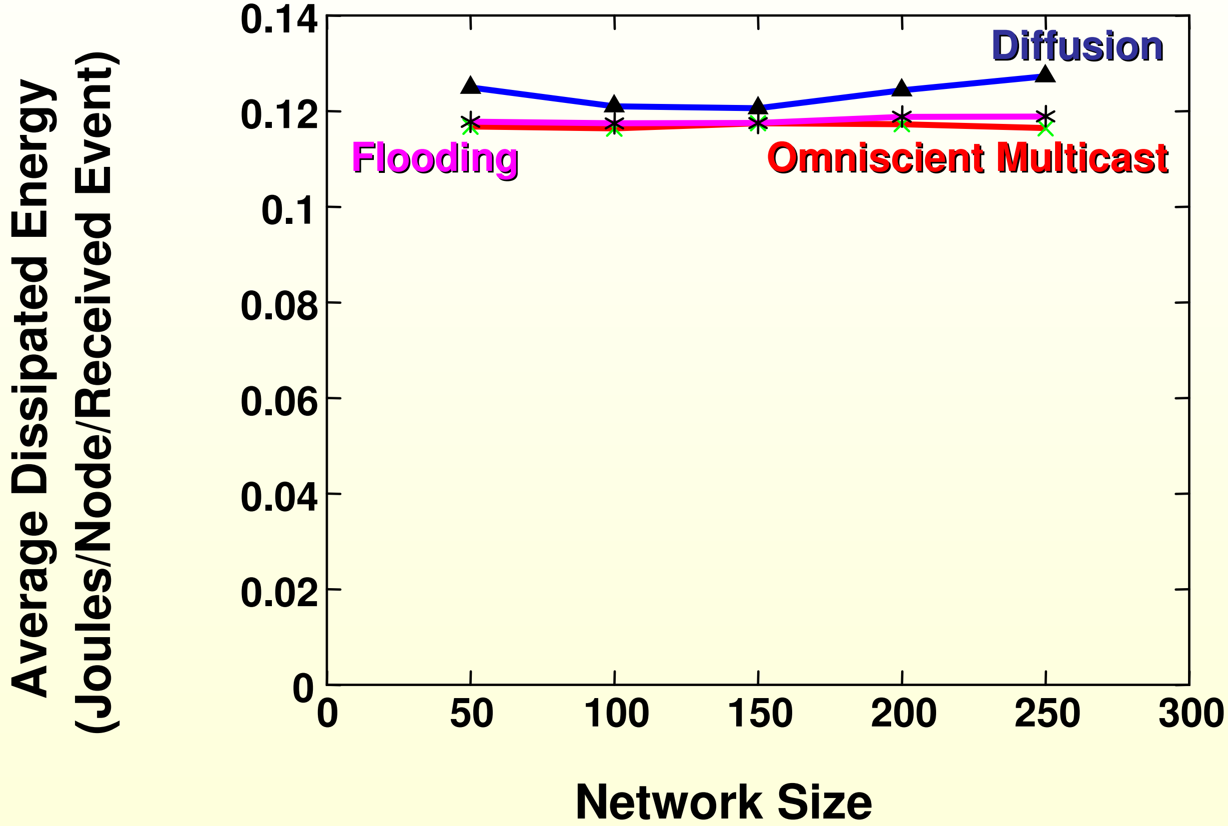
Diffusion Simulation Details

- Simulator: ***ns-2***
- Network Size: 50-250 Nodes
- Transmission Range: 40m
- Constant Density: 1.95×10^{-3} nodes/m^2 (9.8 nodes in radius)
- MAC: Modified Contention-based MAC
- Energy Model: Mimic a realistic sensor radio [Pottie 2000]
  - 660 mW in transmission, 395 mW in reception, and 35 mw in idle mode

Diffusion Simulation

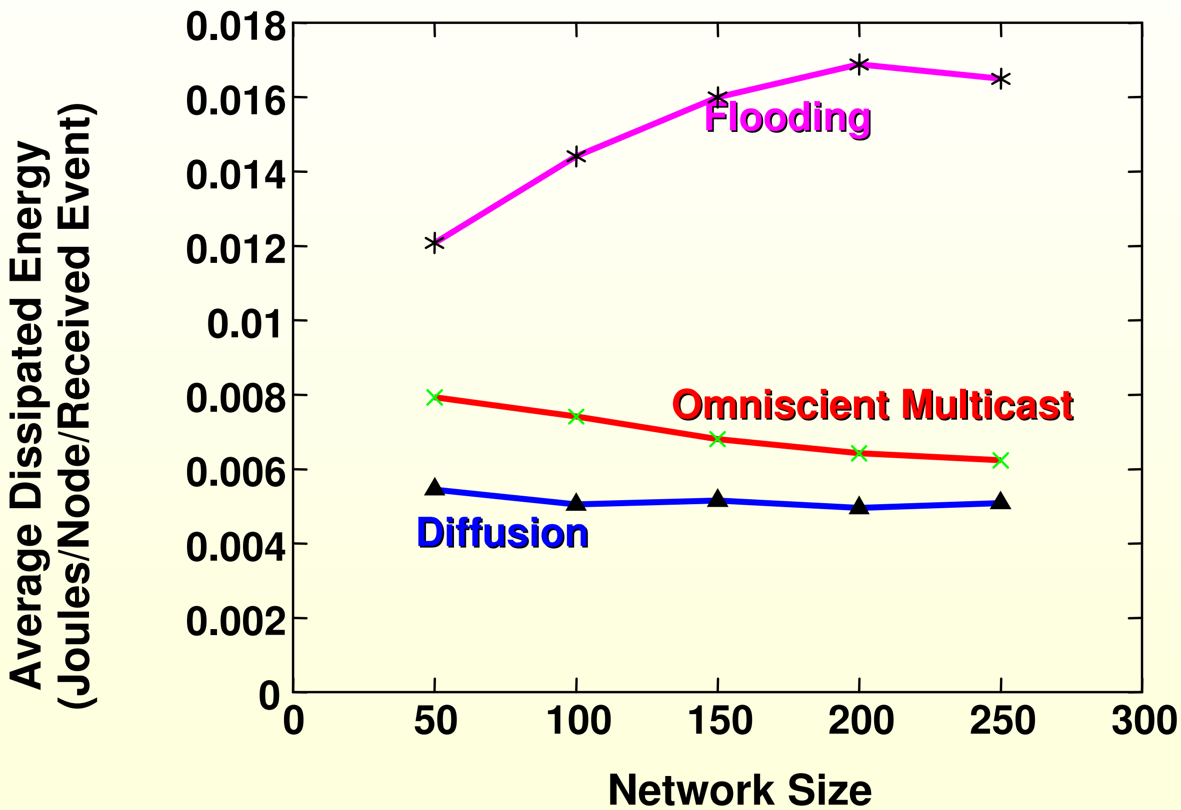
- Surveillance application
  - 5 sources are randomly selected within a 70m x 70m square field
  - 5 sinks are randomly selected across the field
  - High data rate is 2 events/sec
  - Low data rate is 0.02 events/sec
  - Event size: 64 bytes
  - Interest size: 36 bytes
  - All sources send the same location estimate for base experiments

Average Dissipated Energy (**Standard 802.11** energy model)



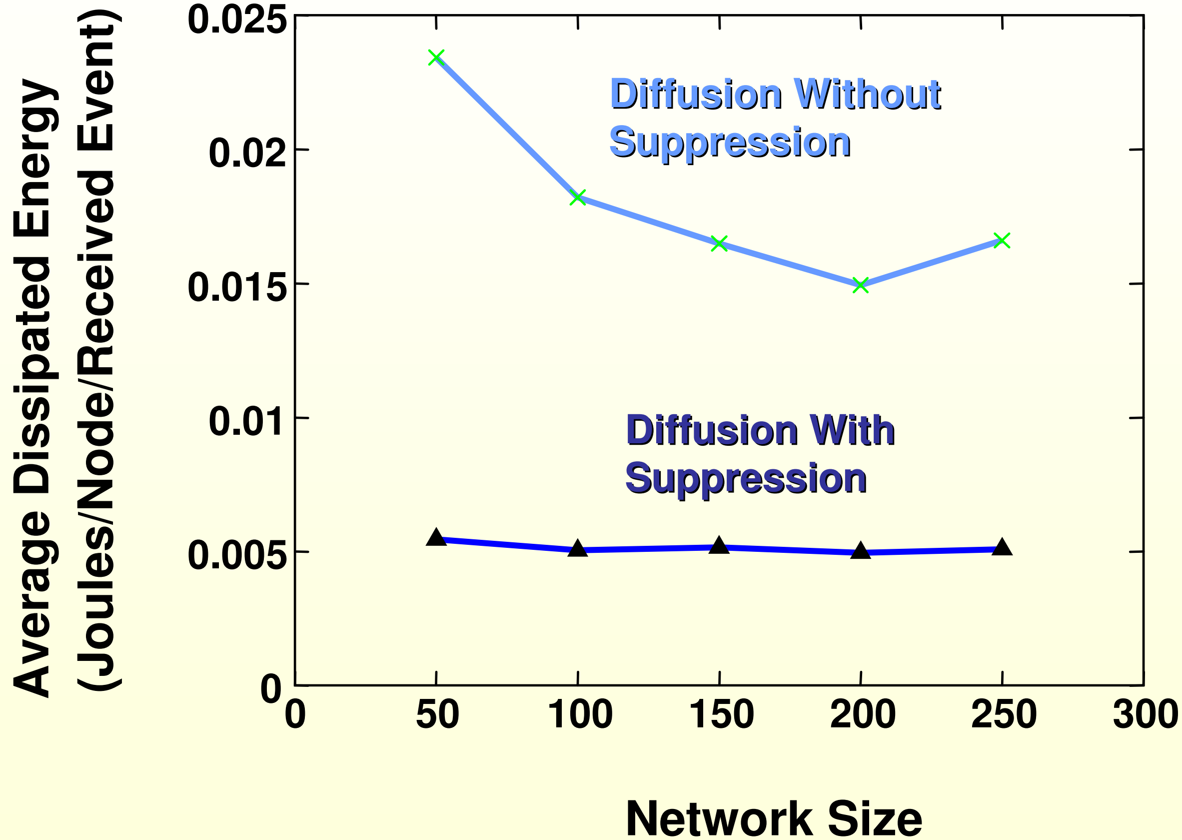
Standard 802.11 is dominated by idle energy

Average Dissipated Energy (**Sensor radio** energy model)



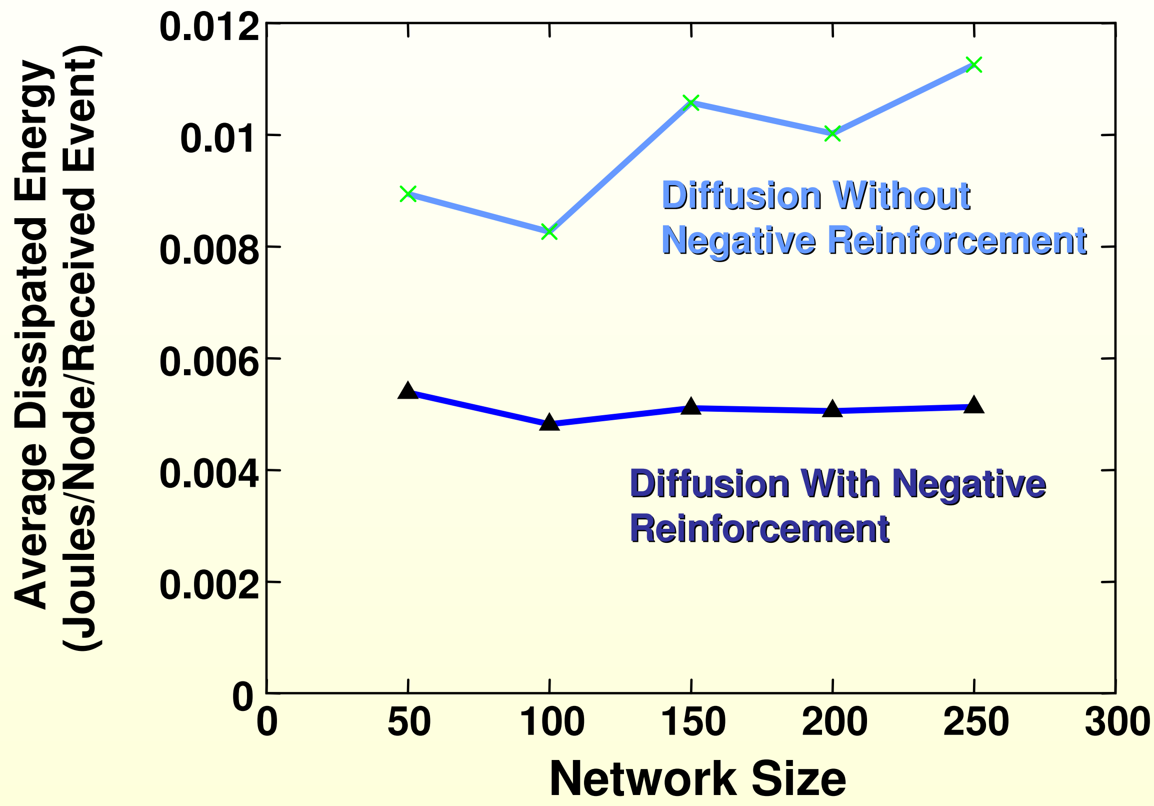
Diffusion can outperform flooding and even omniscient multicast. Why ?

Impact of In-Network Processing



Application-level duplicate suppression allows diffusion to reduce traffic and to surpass omniscient multicast.

Impact of Negative Reinforcement



Reducing high-rate paths in steady state is critical

Summary of Diffusion Results

- Under the investigated scenarios, diffusion outperformed omniscient multicast and flooding
- Application-level data dissemination has the potential to improve energy efficiency significantly
  - Duplicate suppression is only one simple example out of many possible ways.
  - Data aggregation
- All layers have to be carefully designed
  - Not only network layer but also MAC and application level
- More experimentation is needed

Tiny Diffusion

- Implementation of Diffusion on resource constrained UCB motes
  - 8 bit CPU, 8k program memory, 512 bytes data memory
  - Subset of full system
  - Retains only gradients and condenses attributes to a single tag
  - Entire system runs in less than 5.5 KB memory

Contd…

- Tiny OS adds ~3.5 KB and 144 bytes of data (inclusive support for radio and photo sensor
- Diffusion adds ~2k code and 110 bytes of data to tiny OS

Tiny Diffusion Functionality

- Resource constrained
- Limited cache size -- currently 10 entries of 2 bytes each
- Limited ability to support multiple traffic streams. Currently supports five concurrently active gradients

Pull vs. Push Variations

- One could also diffuse data from source, in search of relevant sinks – a completely dual approach
- Or one could try a combination push/pull strategy:
  - pull: sink 2-D, source 0-d
  - push: sink 0-d, source 2-d
  - what about: sink 1-d, source 1-d

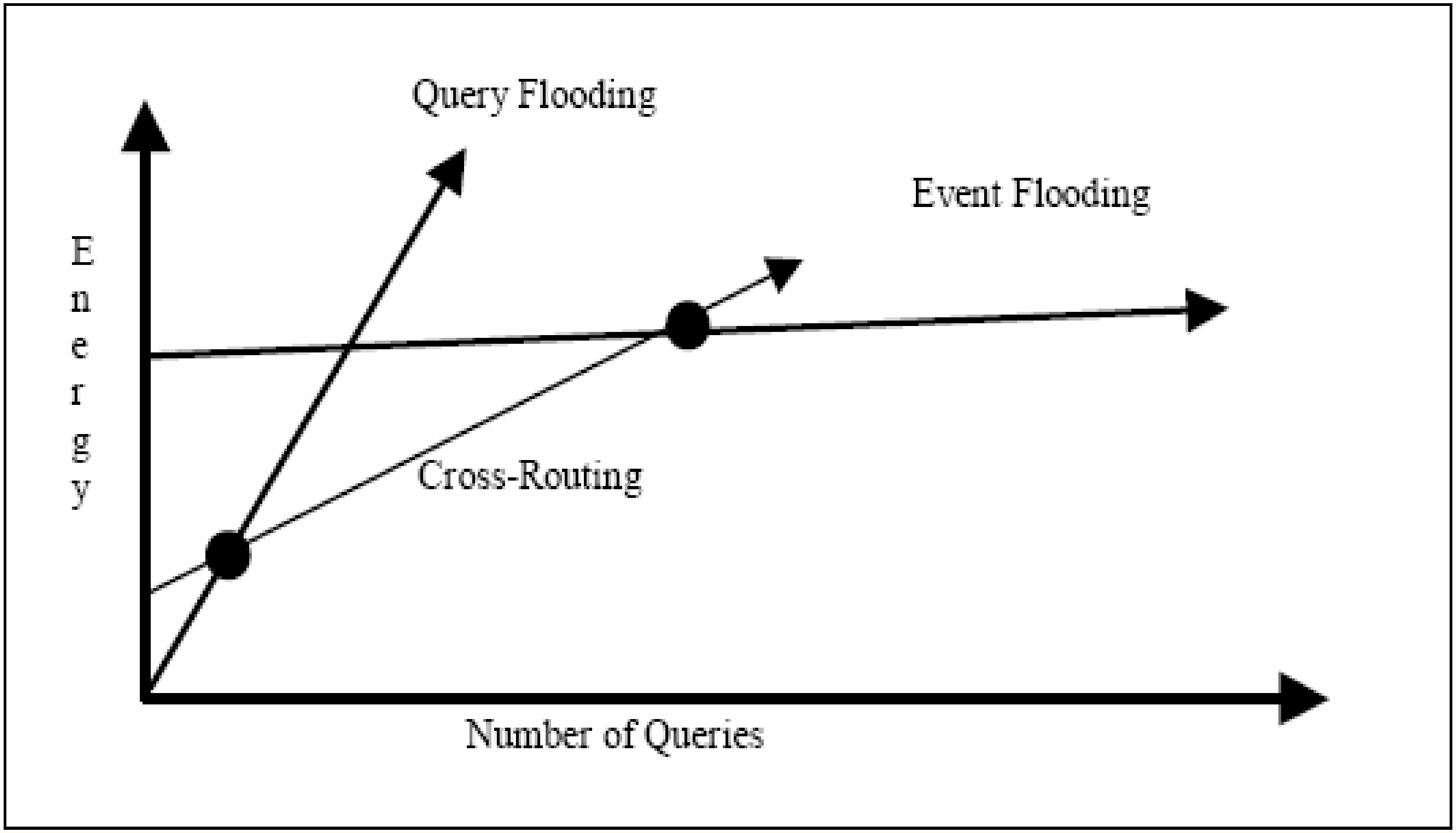
Alternative Methods

- Query flooding
  - Expensive for high query/event ratio
  - Allows for optimal reverse path setup
  - Gossiping schemes can be use to reduce overhead
- Event Flooding
  - Expensive for low query/event ratio
  - there are effective methods for gradient setup
- Note :
  - Both of them provide shortest delay paths

Rumor Routing

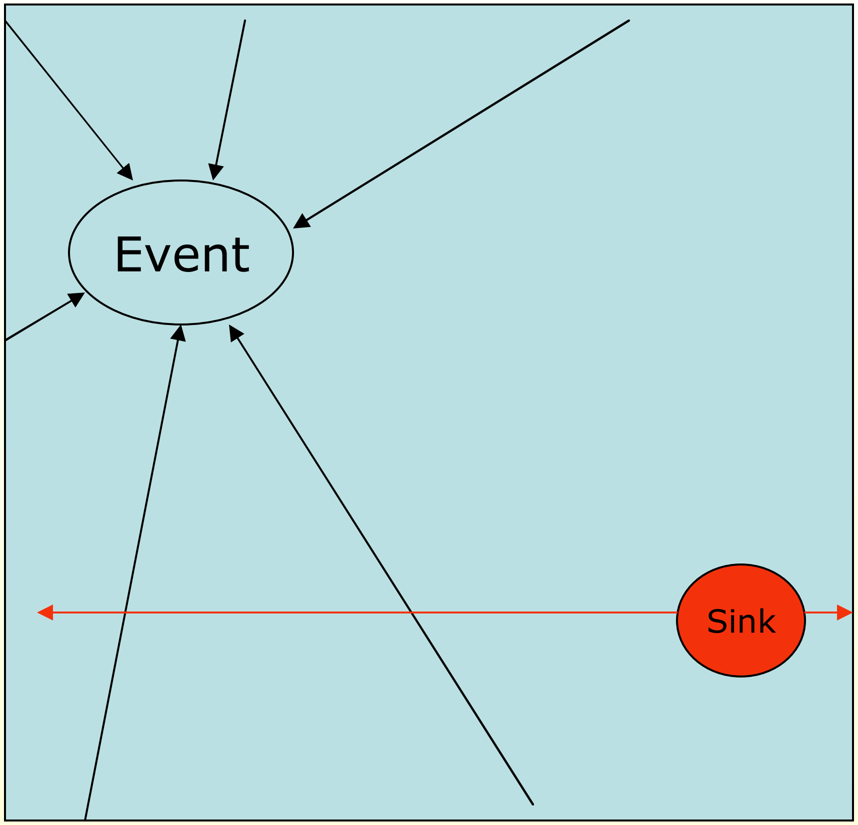
- Designed for query/event ratios between query and event flooding
- Motivation
  - Sometimes a non-optimal route is satisfactory
- Advantages
  - Tunable best effort delivery
  - Tunable for a range of query/event ratios
- Disadvantages
  - Optimal parameters depend heavily on topology (but can be adaptively tuned)
  - Does not guarantee delivery

Rumor Routing



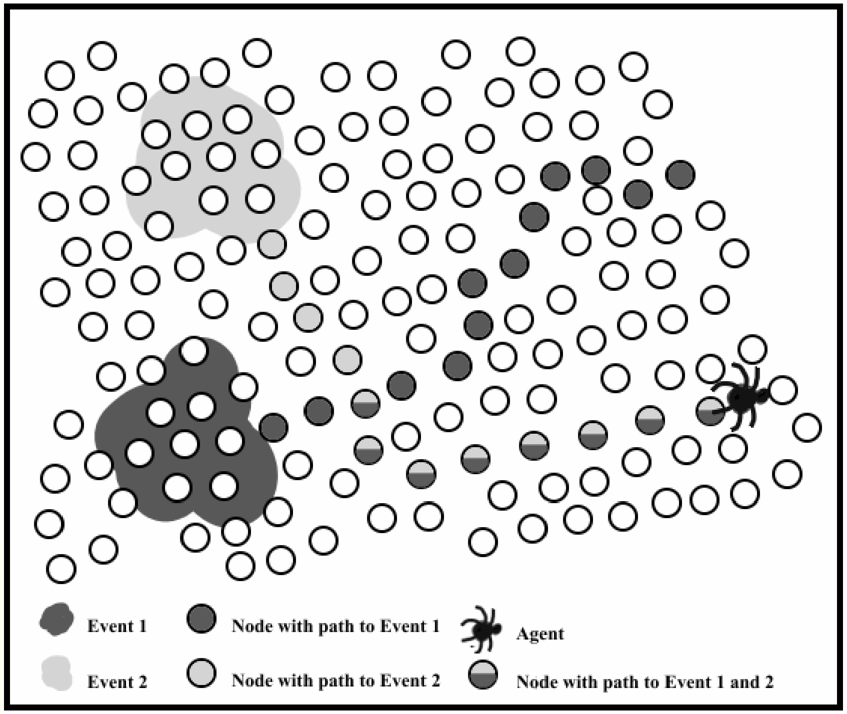
Basis for Algorithm

- Observation: Two lines in a bounded rectangle have a 69% chance of intersecting
- Create a set of straight line gradients from event, then send query along a random straight line from source.



Creating Paths

- Nodes having observed an event send out agents which leave routing info to the event as state in the nodes they pass through
- Agents attempt to travel in a straight line
- If an agent crosses a path of another event, it begins propagates paths to both
- Agents also optimize paths if they find shorter ones.



Algorithm Basics

- All nodes maintain a neighbor list.
- Nodes also maintain a event table
  - When a node observes an event, the event is added to the event table with distance 0.
- Agents
  - Agents are packets that carry local event info across the network.
  - Agents aggregate events as they go.

Agents

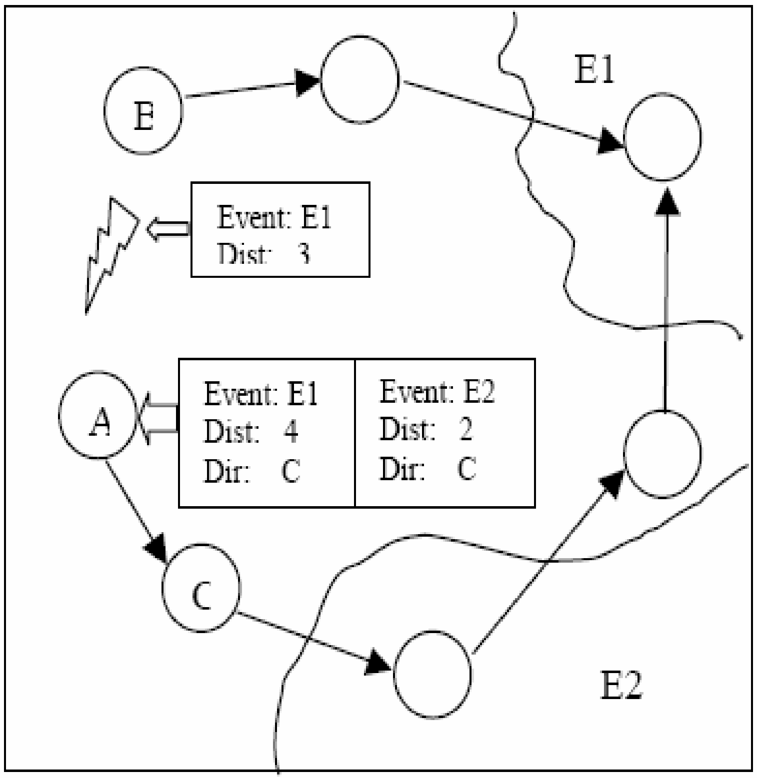


Figure 5

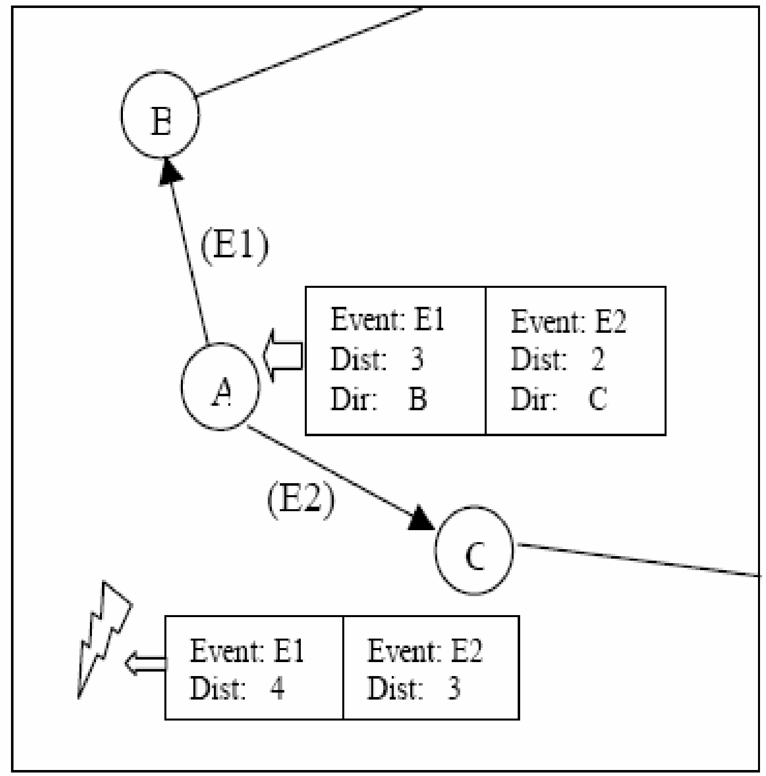


Figure 6

Agent Path

- An agent tries to travel in a “somewhat” straight path.
  - Maintains a list of recently seen nodes.
  - When it arrives at a node, it adds the node’s neighbors to the list.
  - For its next hop, it tries to find a node not in the recently seen list.
  - Avoids loops
  - Important to find a path regardless of “quality”

Following Paths

- A query originates from source, and is forwarded along until it reaches its TTL (time to live)
- Forwarding Rules:
  - If a node has seen the query before, it is sent to a random neighbor
  - If a node has a route to the event, forward to neighbor along the route
  - Otherwise, forward to random neighbor using straightening algorithm

Energy Comparison

- Rumor Routing (1000 queries)
  - Es + Q*(Eq + N*(1000-Qf)/1000)
  - Es = avg. energy to set up path
  - Eq = avg. energy to route a query
  - Qf = successful queries
  - Q queries are routed
- Query Flooding
  - Q*N
- Event Flooding
  - E*N

Simulation Scenario

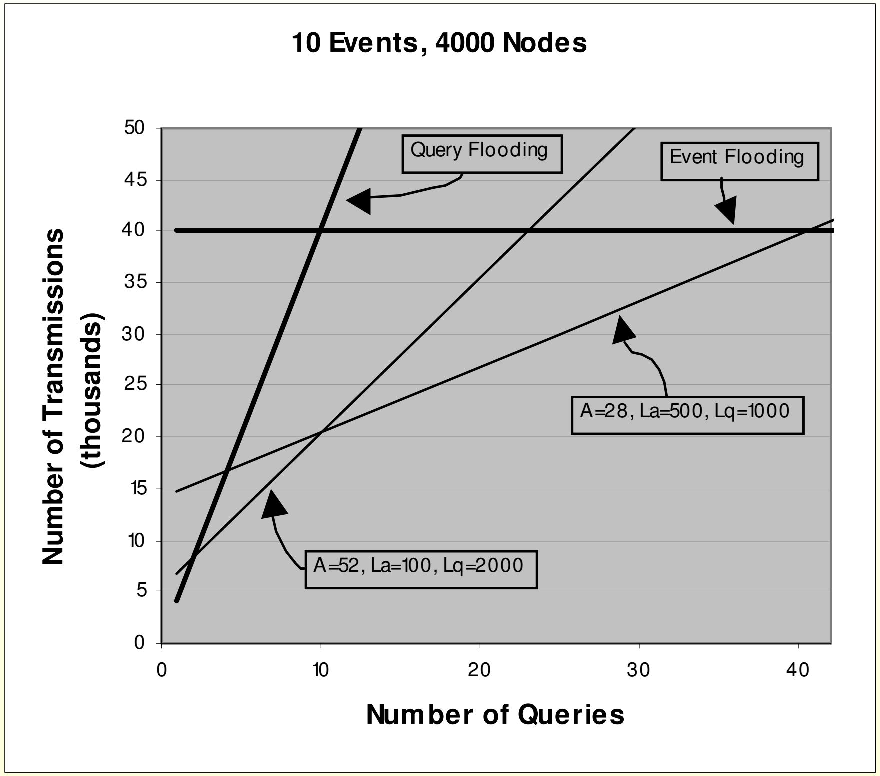
- Simple radial propagation model with symmetric reliable transmission (r=5)
- Dense network of nodes (3000, 4000, 5000 in field of 200x200m^2)
- Simultaneous circular events of radius 5m (10, 50, 100)
- Varied parameters to find optimal ranges
  - Number of agents per event
  - Agent TTL
  - Query TTL

Simulation Results

- Bad : Agent TTL 100, number of agents around 25.
- Large value of number of agents (around 400) had high setup cost but better delivery rate, so lower average energy consumption.
- Best Result
  - Agents = 31
  - Agent TTL 1000
  - 98.1 % queries delivered
  - energy spent 1/20-th of a network flood.

Simulation Results

- Assume that undelivered queries are flooded
- Wide range of parameters allow for energy saving over either of the naïve alternatives
- Optimal parameters depend on network topology, query/event distribution and frequency
- Algorithm was very sensitive to event distribution



Fault Tolerance

- After agents propagated paths to events, some nodes were disabled.
- Delivery probability degraded linearly up to 20% node failure, then dropped sharply
- Both random and clustered failure were simulated with similar results

Some Thoughts

- The effect of event distribution on the results is not clear.
- The straightening algorithm used is essentially only a random walk … can something better be done?
- The tuning of parameters for different network sizes and different node densities is not clear.
- There are no clear guidelines for parameter tuning, only simulation results in a particular environment.

Information Brokerage Using Network Storage

Storage in a Sensor Network

- In some applications, continuously streamed data from sources is not required
- It is sufficient to save a summarized form of the data, for later retrieval
- But where should this data be stored? And how can it be retrieved?
- More about this in a few weeks ...

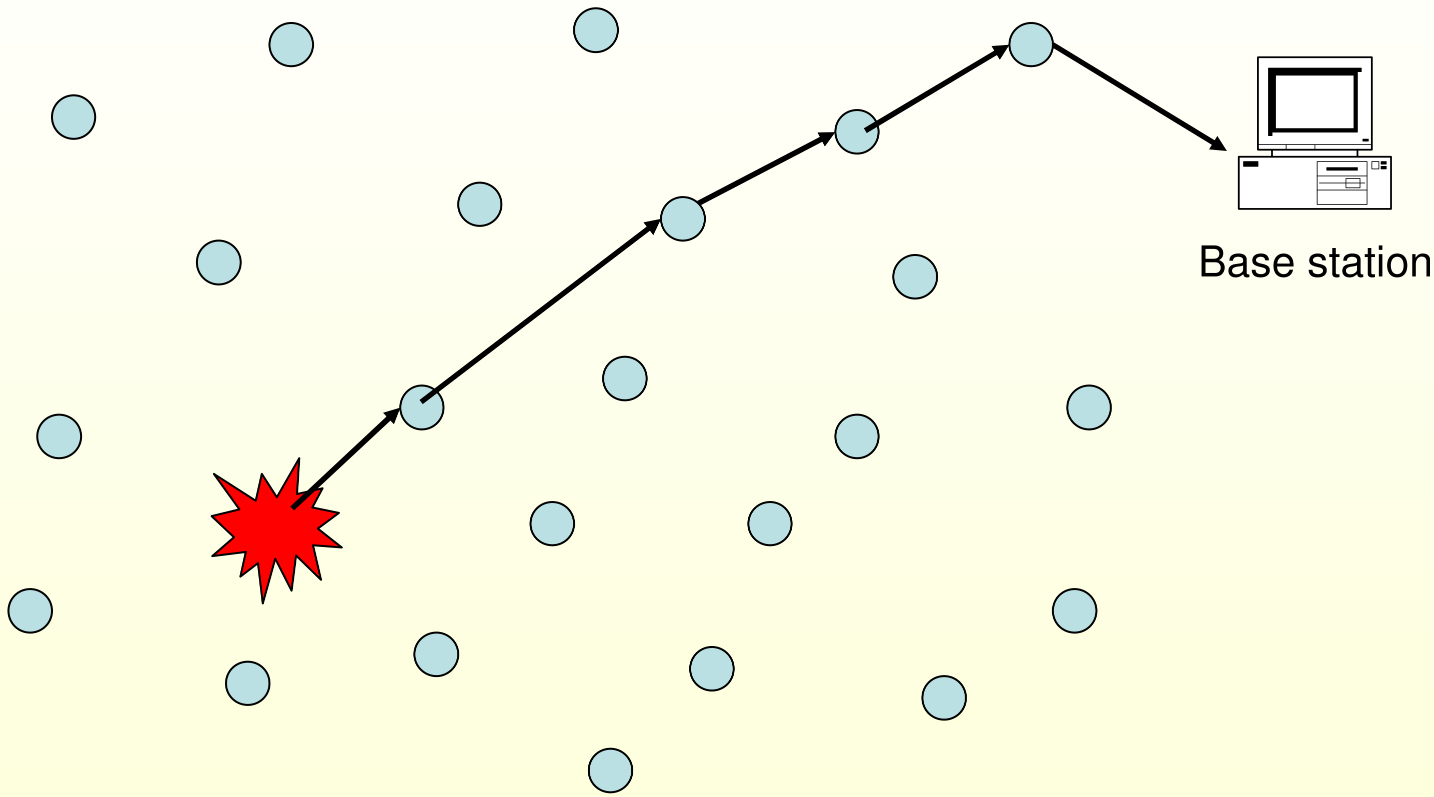
Observations/Events/Queries

- **Observations**
  - Low-level output from sensors
- **Events**
  - Constellations of low-level observations, interpreted as higher-level events or activities
  - E.g. fire, intruder
- Clients use **Queries** to elicit event information from sensor network
  - E.g.: Locations of fires in the network
  - E.g.: Images of intruders detected

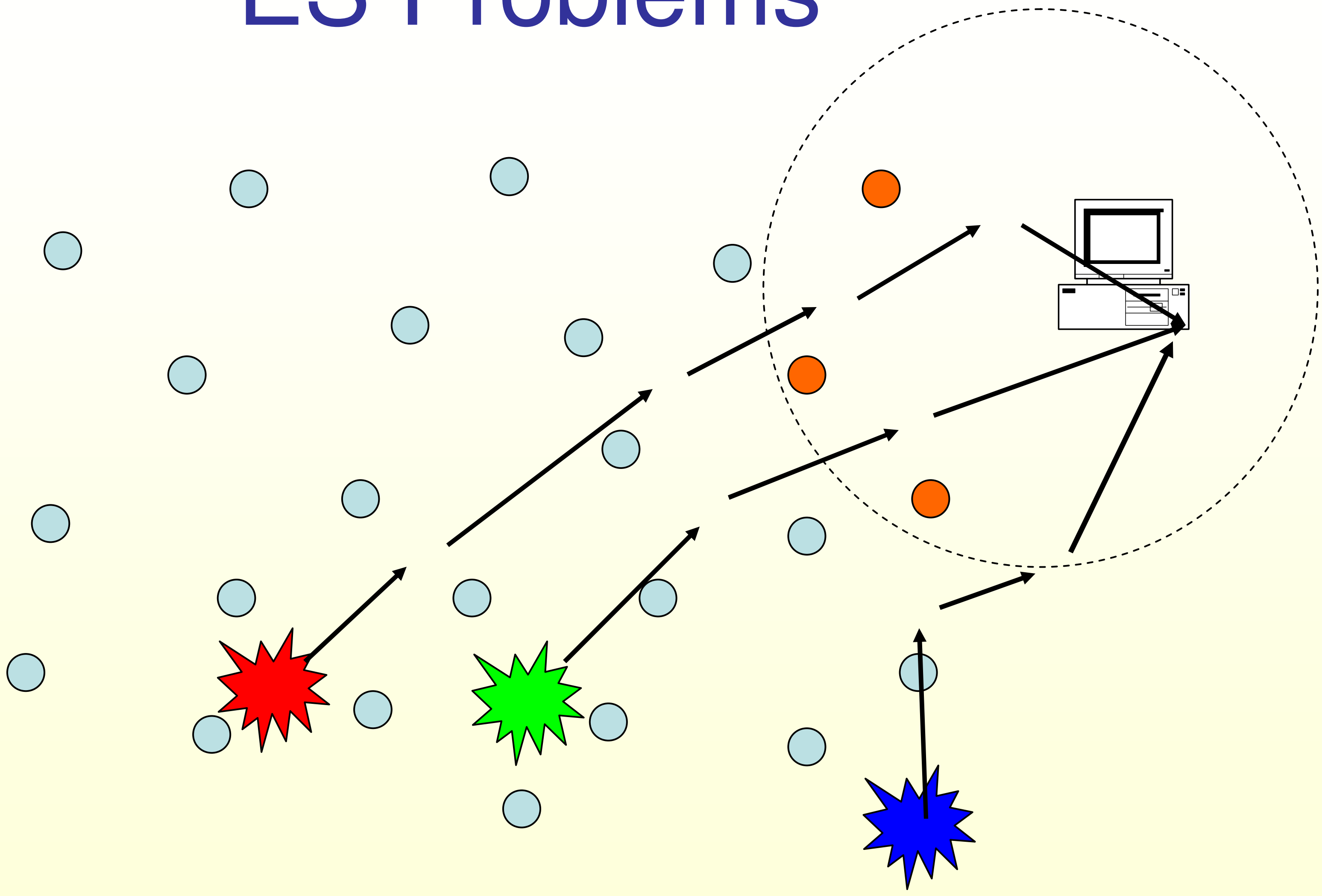
Possible Approaches

- External Storage (ES)
- Local Storage (LS)
- Data-Centric Storage (DS)

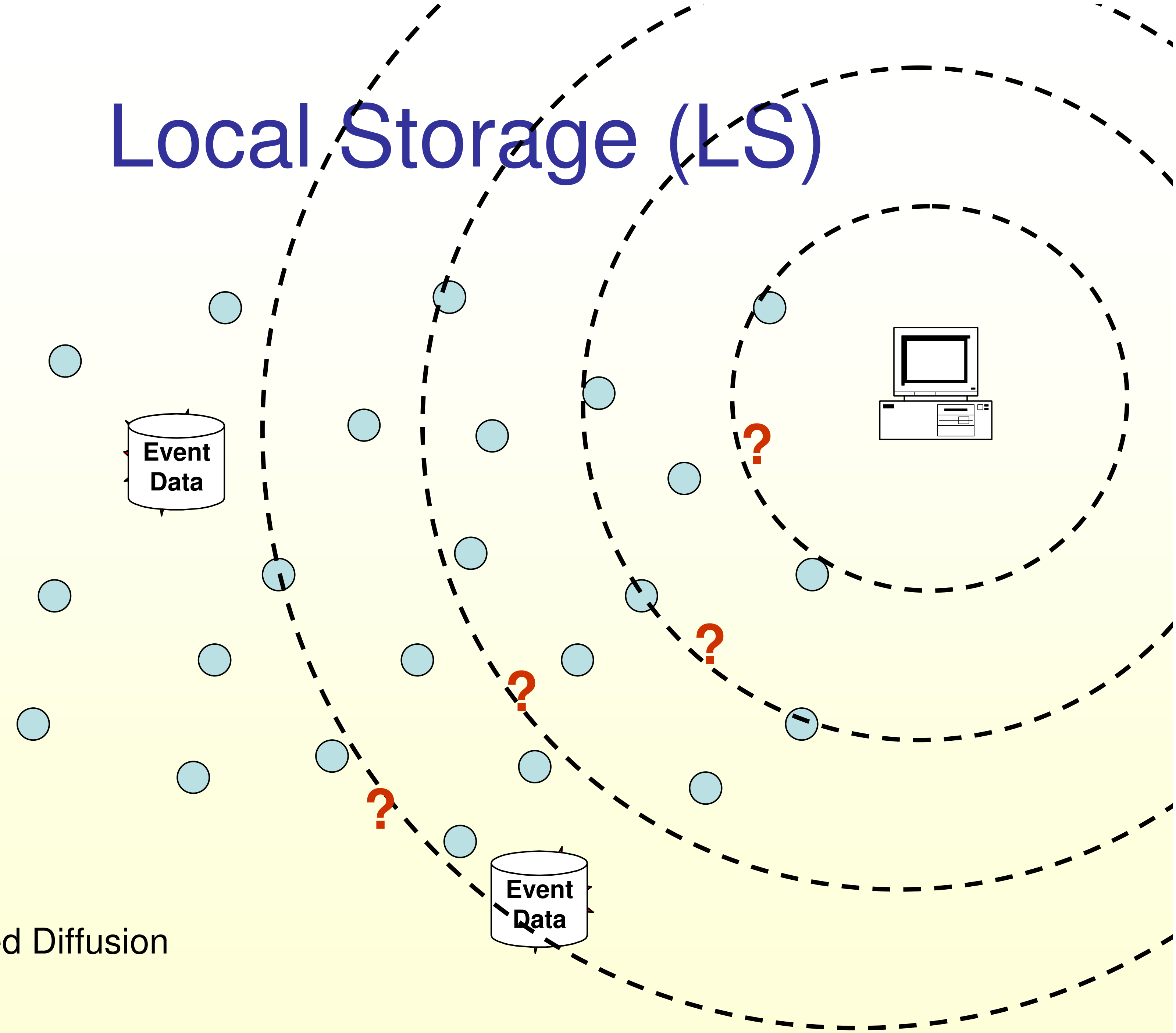
External Storage (ES)



ES Problems

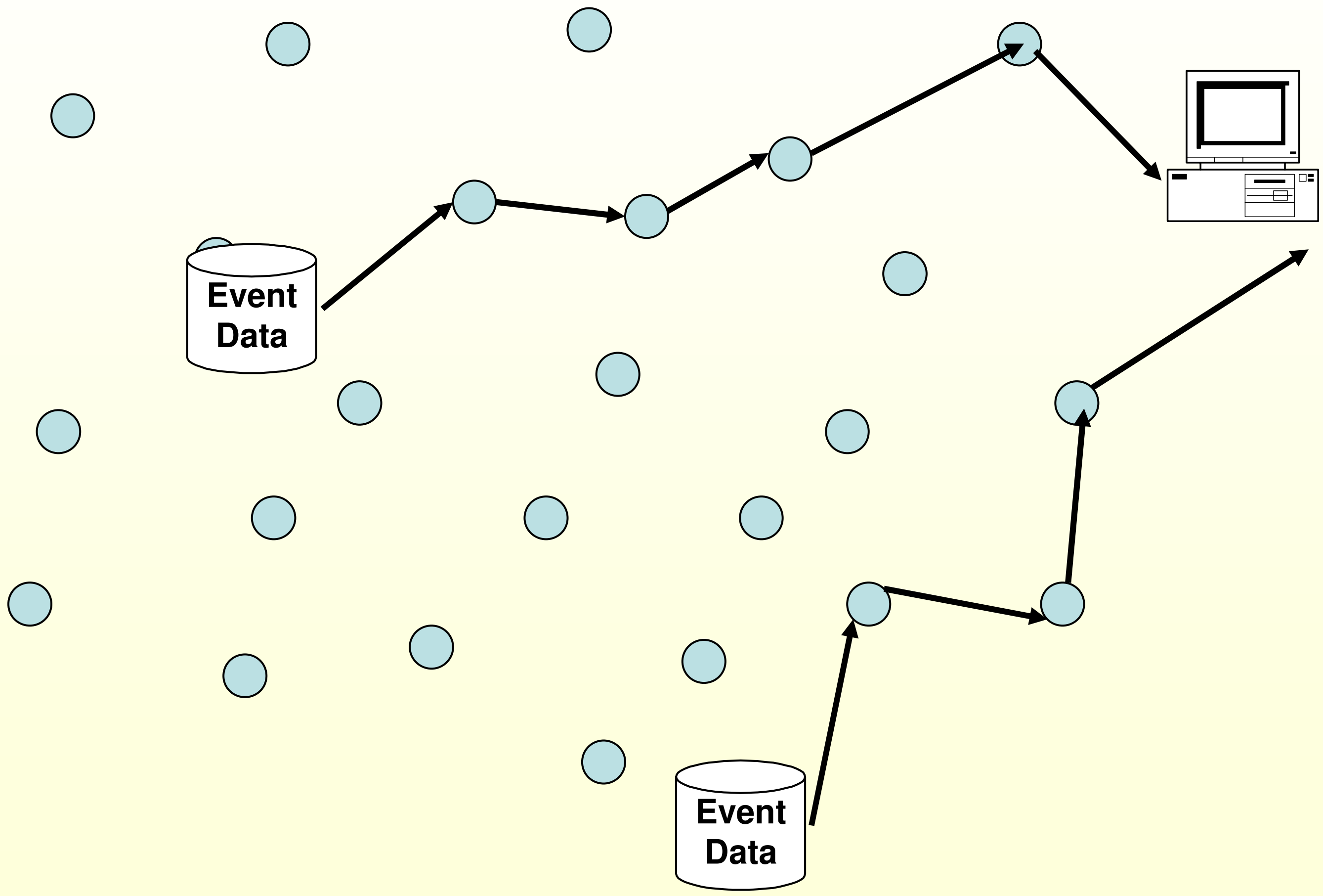


Local Storage (LS)



Directed Diffusion

Local Storage (LS)



Data-Centric Storage (DCS)

- Data-Centric: data is named by attributes
- Event data is stored, by name, at home nodes; home nodes are selected by the named attributes
- Queries also go to the home nodes to retrieve the data (instead of to the nodes that detected the events)
- Home nodes are determined by a hash function + GPSR

Algorithms Used by GHT

- Geographic hash table uses GPSR for routing

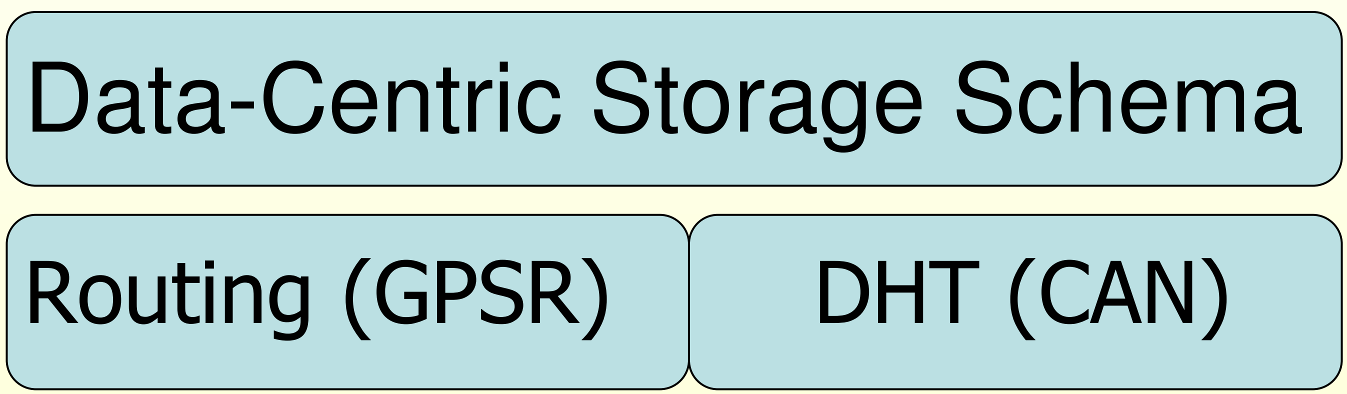
(Greedy perimeter stateless routing)

- PEER-TO-PEER look up system

(data object is associated with key and each node in the system is responsible for storing a certain range of keys)

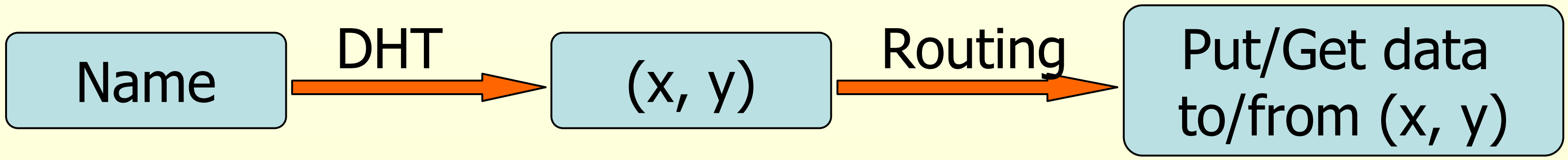
The Big Picture

- Based on geographic routing (Karp) and P2P lookup algorithm (Ratnasamy)



Distributed Hash Table (DHT)

- **`void Put(key,value)`**
  - Stores **`value`** in home node of the sensor network, according to attribute **`key`**
- **`Value Get(key)`**
  - Retrieve **`value`** from home node of the sensor networks according to **`key`**

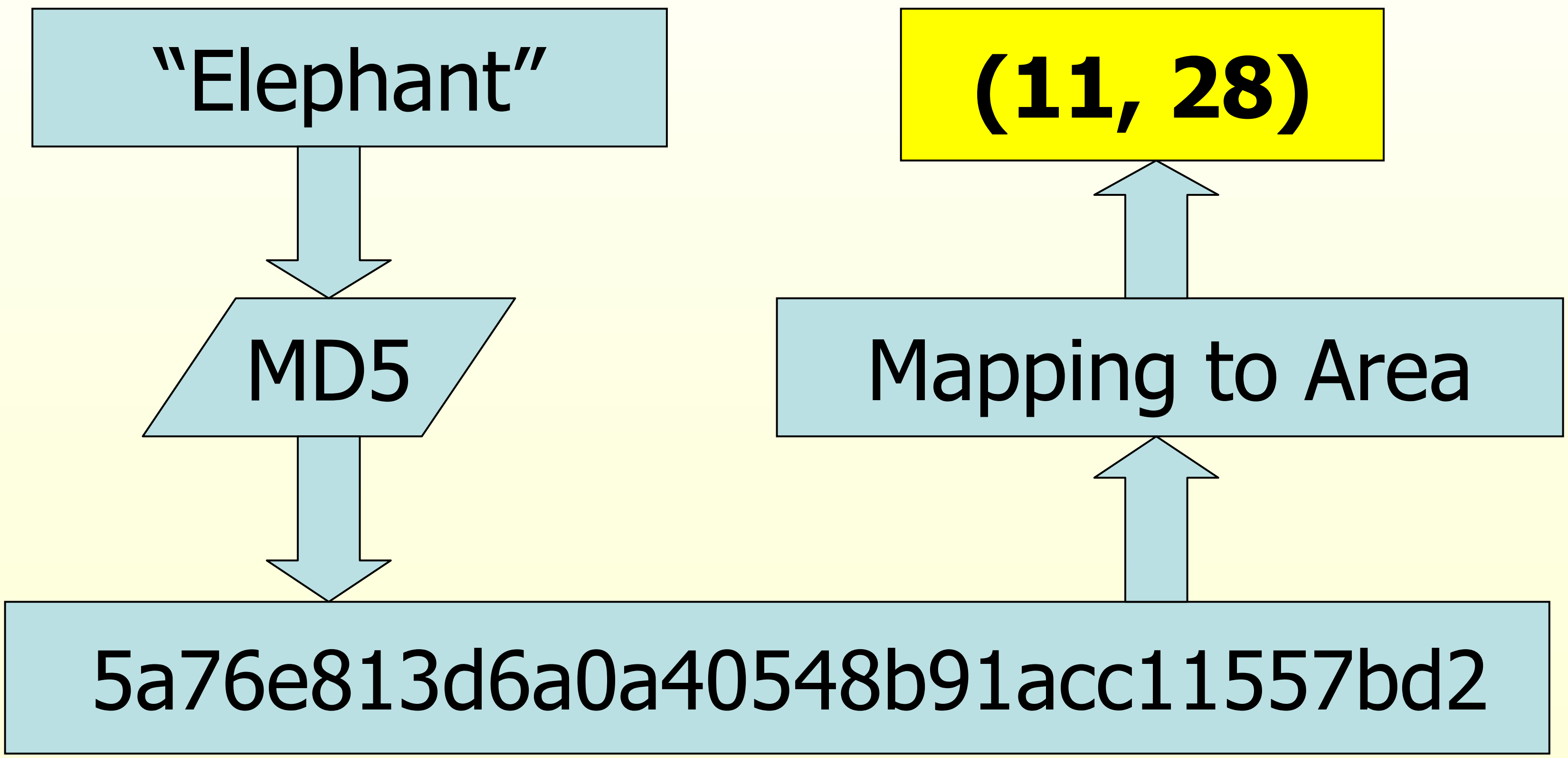


Properties of DHT

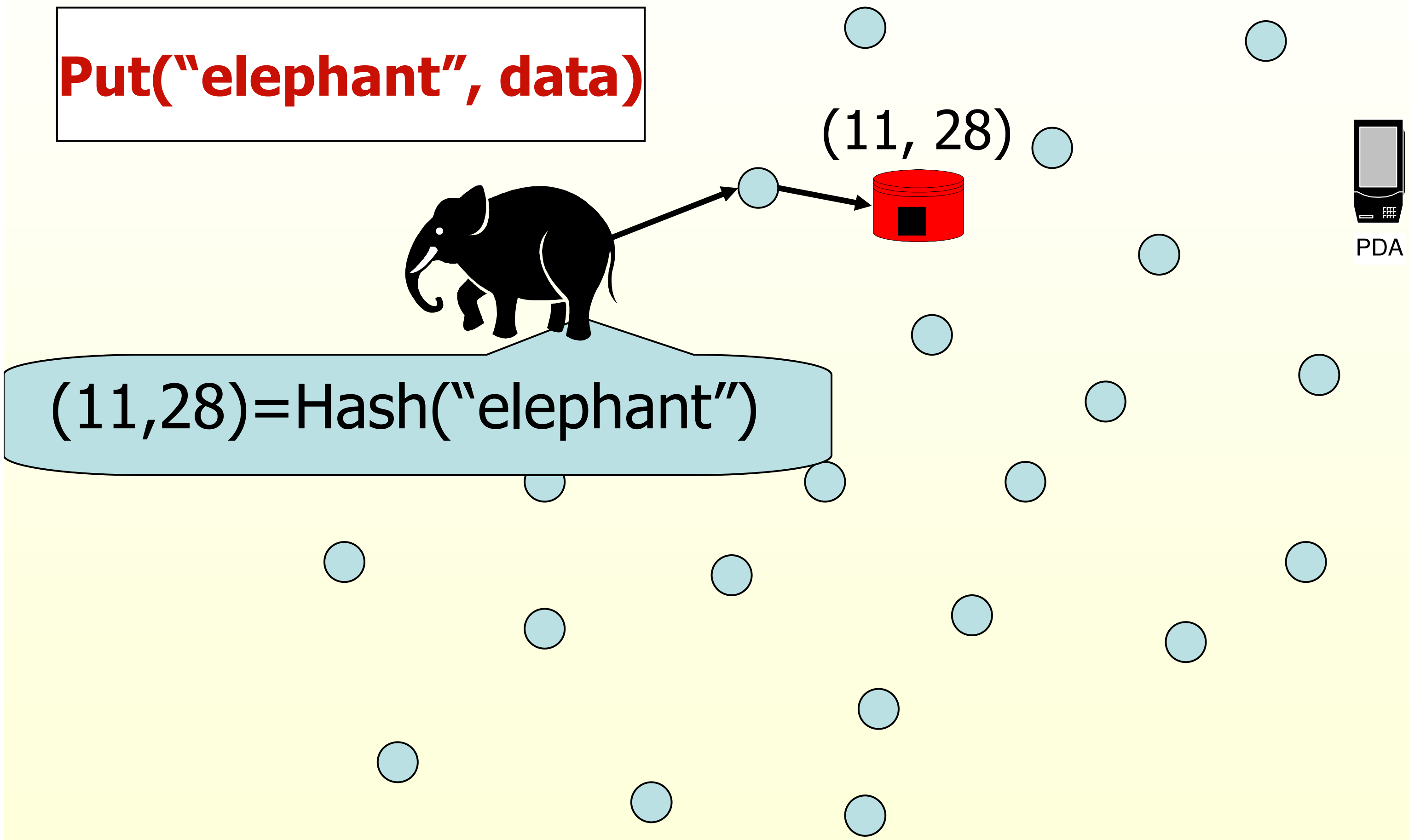
- Uses a distributed hash function
  - Hash function is known to all nodes
  - Every home node takes care of roughly the same amount of event types
  - Evenly distributed geographically
- Candidate: Message Digest Algorithms
  - Such as SHA-1, MD5

DHT - Example

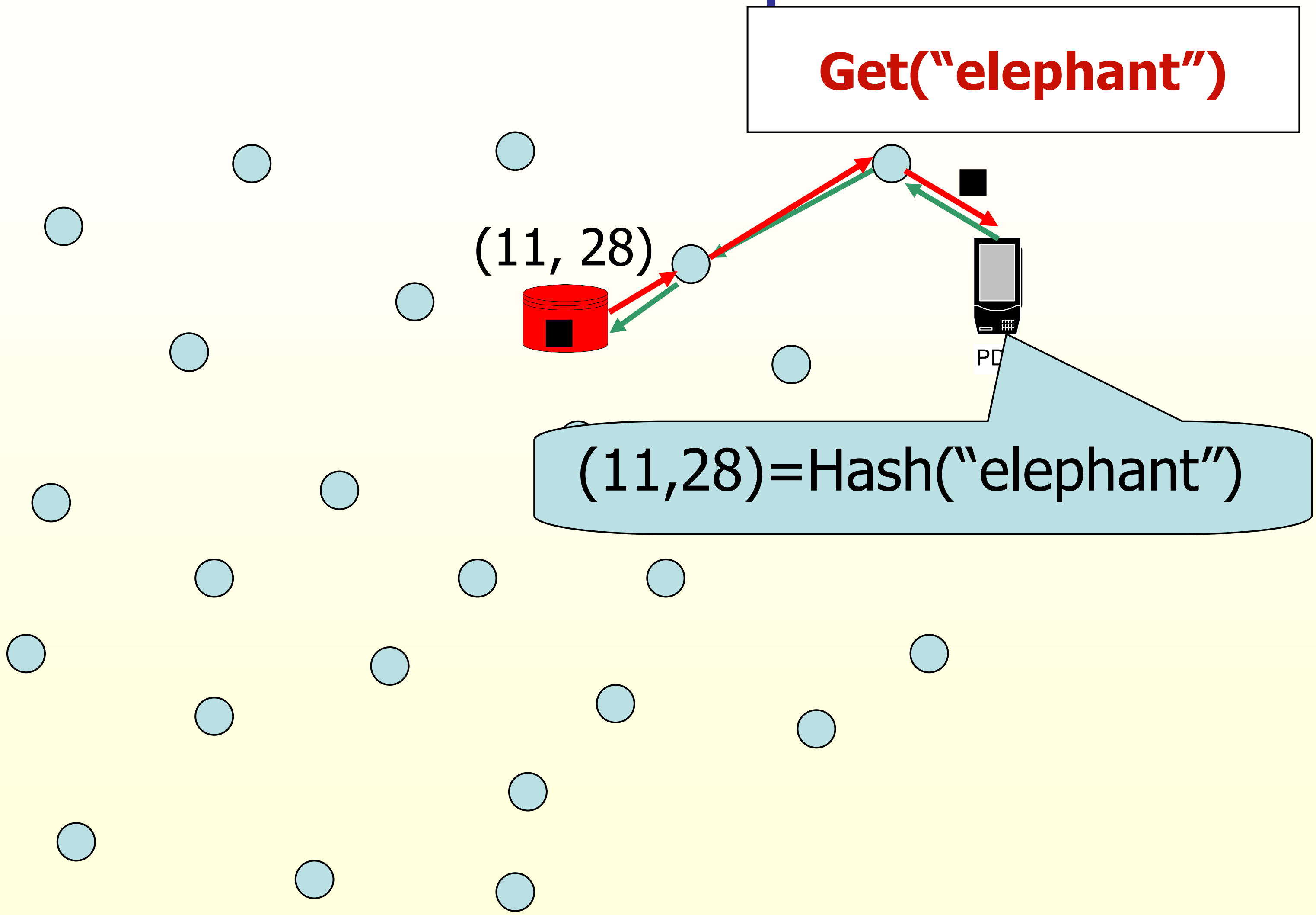
- Example



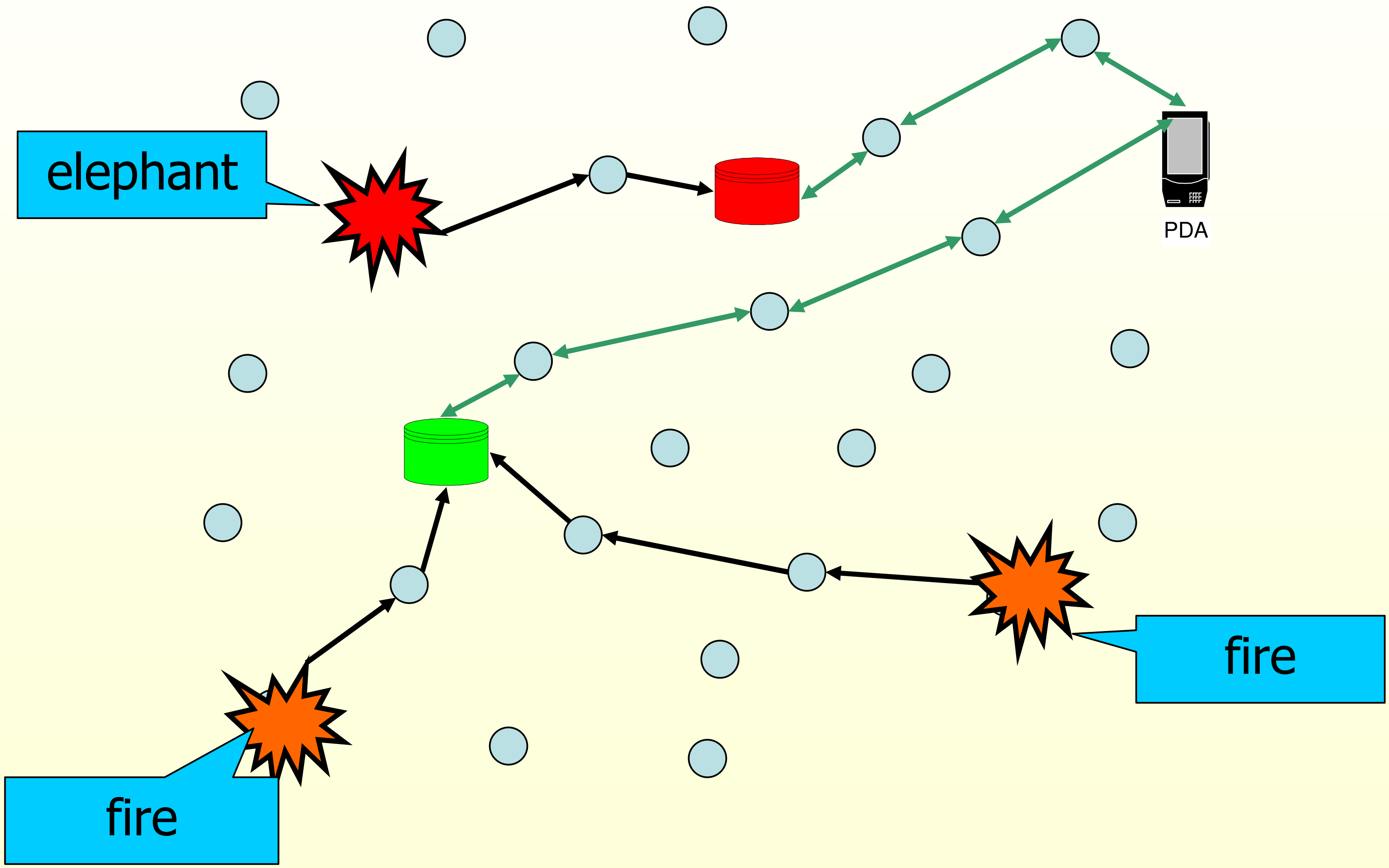
DCS – Example Revisit



DCS – Example

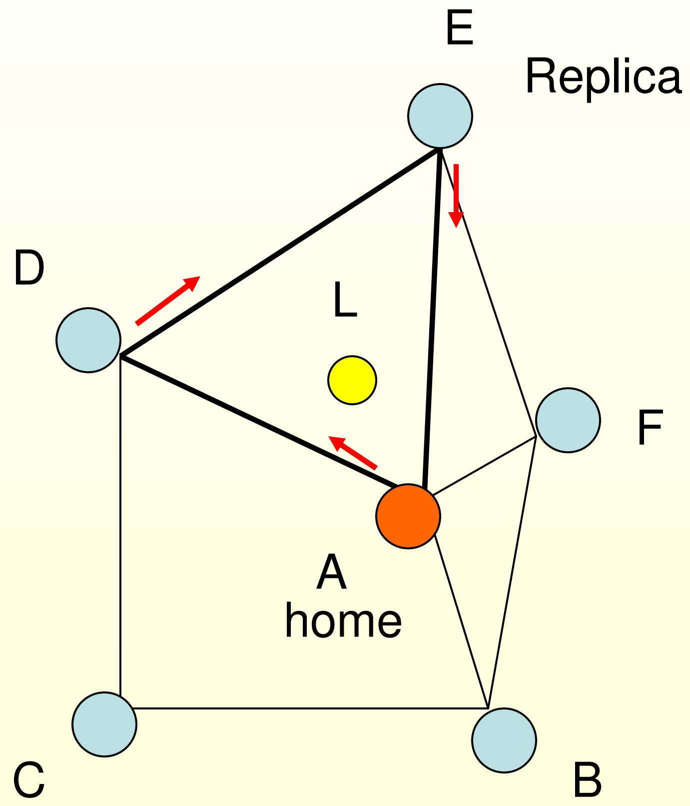


DCS – Example



Home Node and Home Perimeter

- In GHT packet is not addressed to specific node *but only to a specific location in the field*
- The packet will circle around the face of the GPSR face containing the destination location
- The packet will traverse the entire perimeter that encloses the destination and eventually be consumed at the home node (the node closest to destination) – and that perimeter is known as the home perimeter



Problems with DCS

- Not robust enough
  - Home nodes could fail
  - Nodes could move (new home node?)
- Not scalable
  - Home nodes could become communication bottlenecks
  - Storage capacity of home nodes

Conclusions

- Brokerage between information providers and seekers is a fundamental problem in wireless sensor networks
- Reactive protocols are best, to accommodate dynamics both in the phenomena being monitored, as well as in the network itself
- Both push and pull paradigms apply, and various combinations
- In-network storage can provide rendez-vous points between data producers and consumers

The End