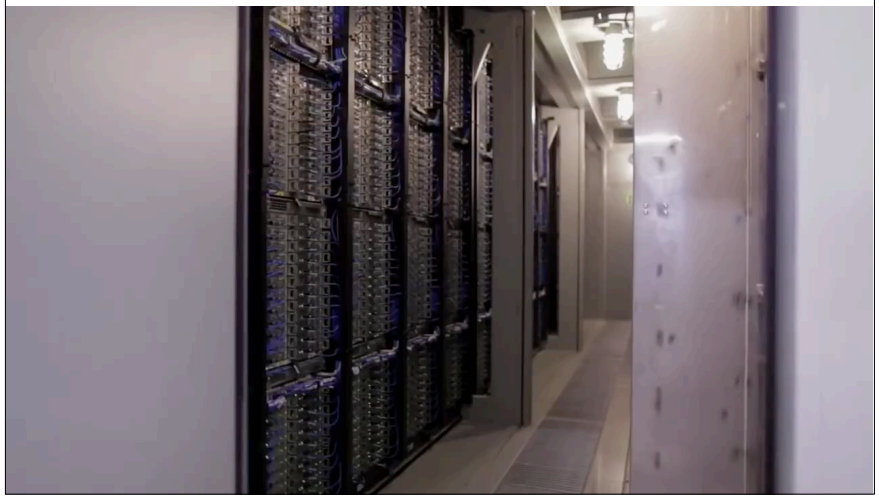
Distributed Computing Models

Sébastien Tixeuil
sebastien.tixeuil@lip6.fr

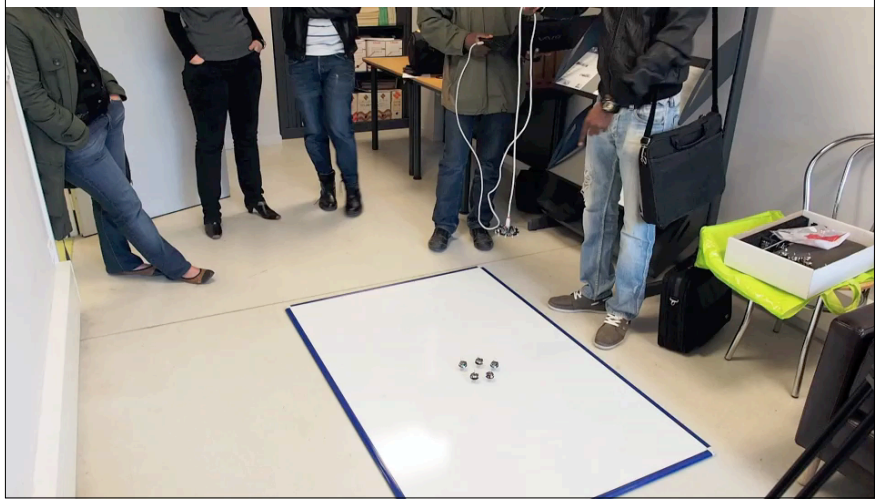
Three Domains

Static Networks



Passively Mobile Networks

Actively Mobile Networks

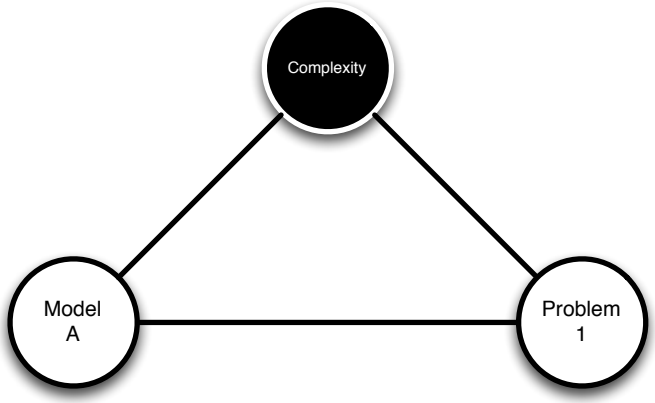


5

Three Approaches

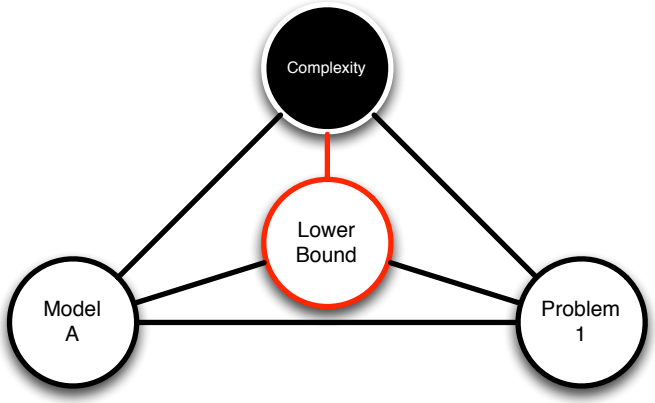
6

Complexity-driven



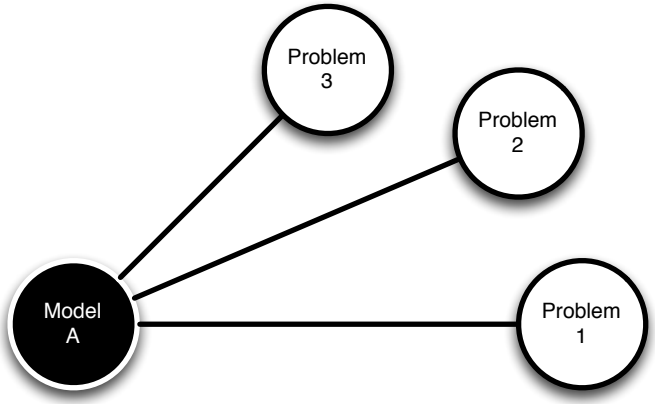
7

Complexity-driven



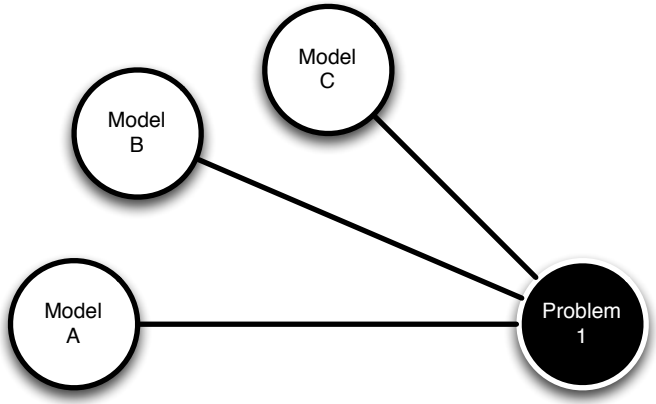
8

Model-driven



9

Problem-driven



10

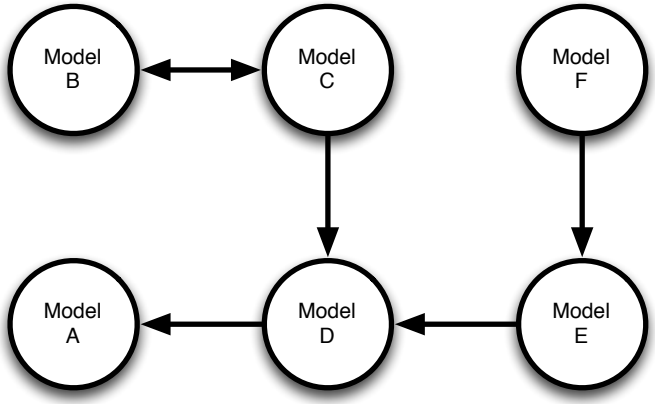
- Landing gear's frozen
- I lost my radar
- I'm out of fuel
- Lost a wing
- Lost the other one

- looking good
- right for line-up
- doing fine
- Ok you can do it

11

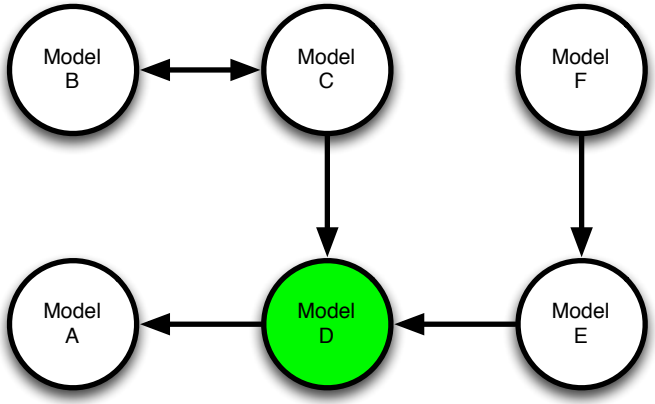
12

A Map of Models



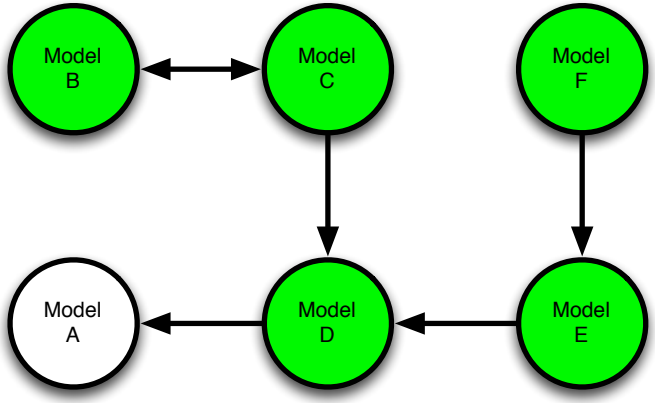
13

A Map of Models



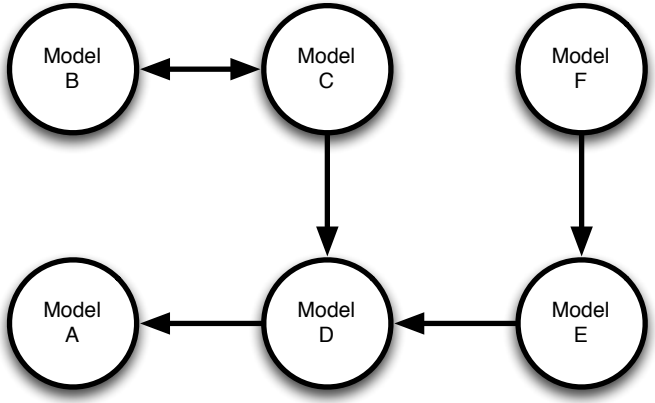
14

A Map of Models



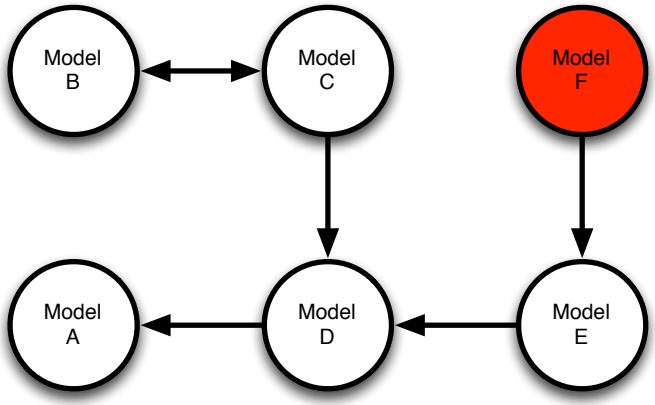
15

A Map of Models



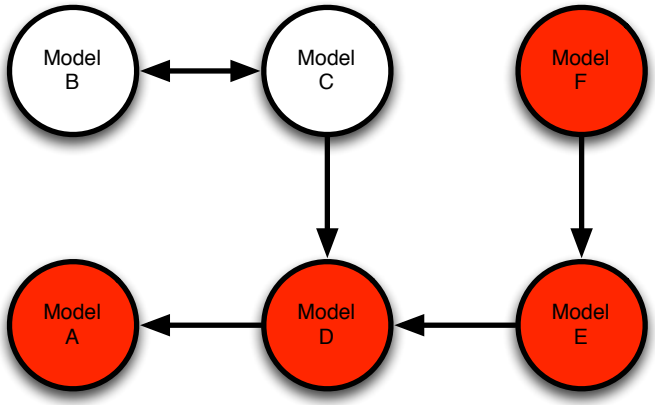
16

A Map of Models



17

A Map of Models



18

Static Networks

19

Parallel Computing

20

Parallel Computing

Parallel Computing

Parallel Computing

- Tractable Sequential Problems
- Homogeneity
- Synchrony
- Reliable
- Focus on **Efficiency**

Distributed Computing

Distributed Computing

Distributed Computing

Distributed Computing

Distributed Computing

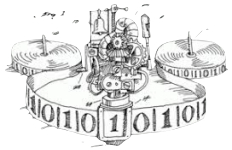
- Intrinsically distributed problems
- Heterogeneity
- Asynchrony
- Unreliable
- Focus on **Computability and Complexity**

Distributed Computing

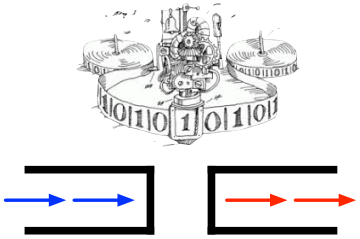
A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Leslie Lamport

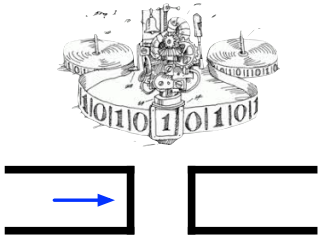
Distributed Computing Elements



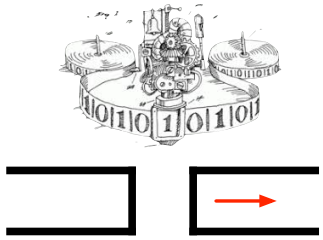
Distributed Computing Elements



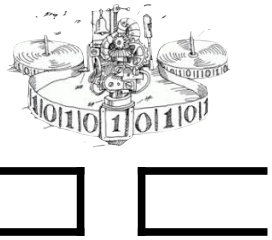
Distributed Computing Processor Actions



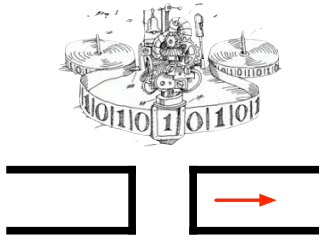
Distributed Computing Processor Actions



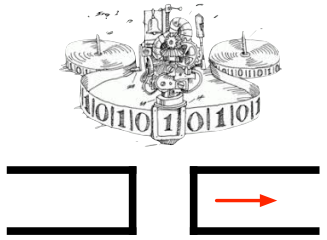
Distributed Computing Processor Actions

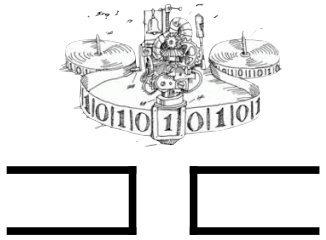


Distributed Computing Processor Actions

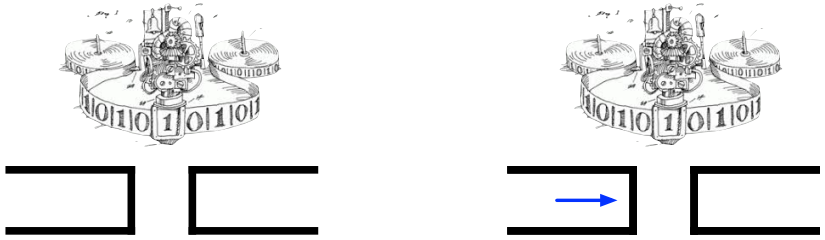


Distributed Computing Link Actions





Distributed Computing Link Actions



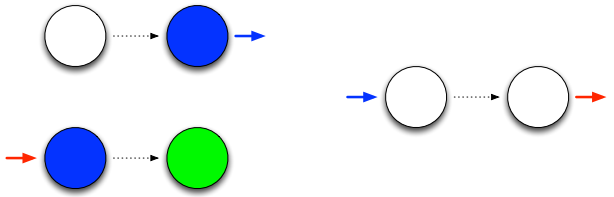
Asynchronous Distributed Execution

- **Sequence** of « processor or link » **actions**
- (***liveness***) Each processor executes an infinite number of actions (or terminates)
- (***liveness***) Each enabled link action eventually occurs

Client-Server

- *Initially*:
  - **Send** Request **to** Server
- *Upon receipt of Response from Server*:
  - **Terminate**
- *Upon receipt of Request from Client*:
  - **Send** Response **to** Client

Client-Server



Client-Server

Client-Server

Client-Server

Client-Server

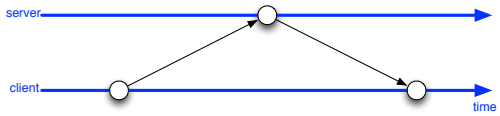
Client-Server

45

Client-Server

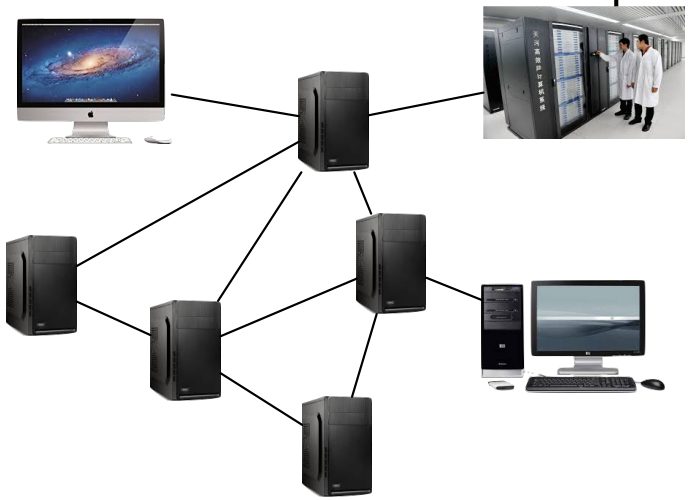
46

Space-Time Diagram



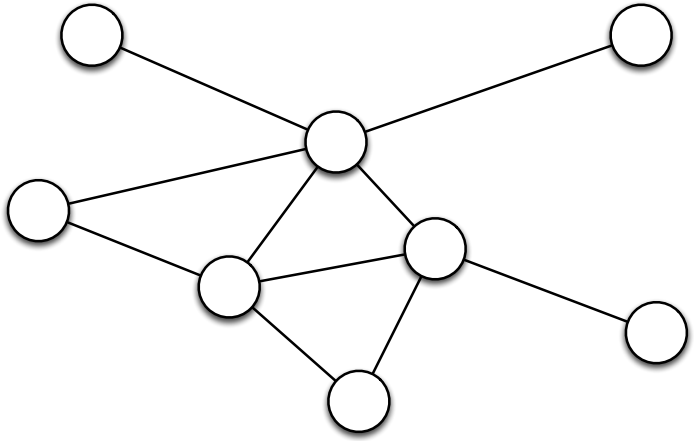
47

Communication Graph



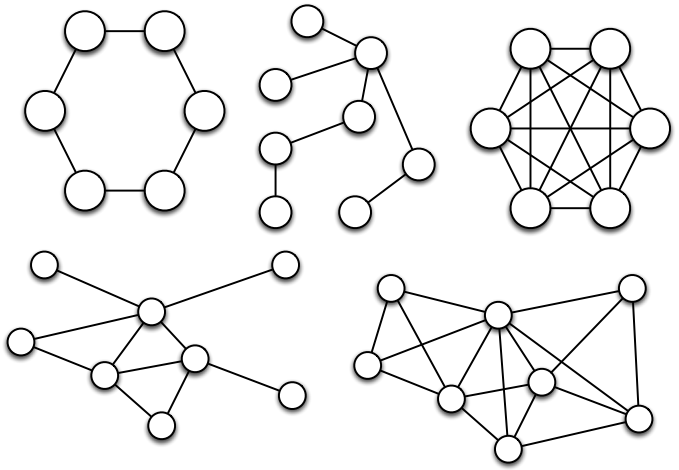
48

Communication Graph



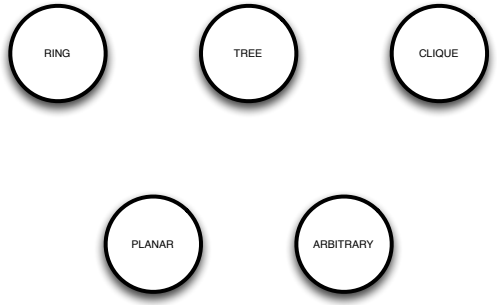
49

Communication Graphs



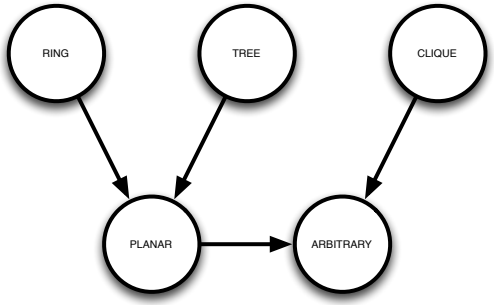
50

Communication Graph Models



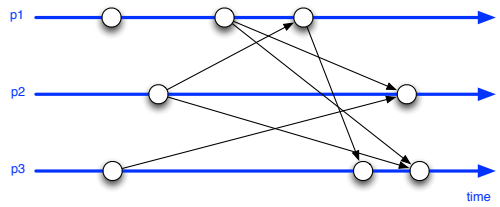
51

Communication Graph Models



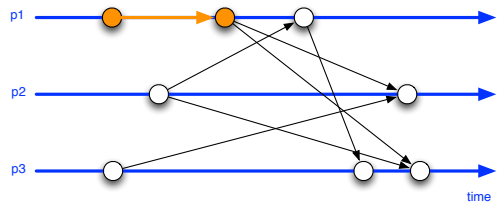
52

Space-Time Diagram



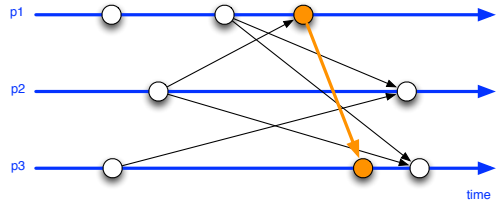
53

Happens Before



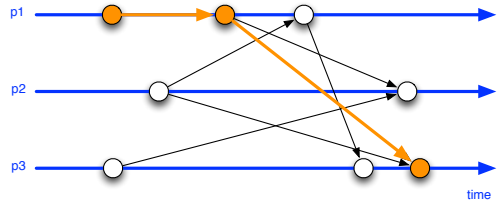
54

Happens Before



55

Happens Before

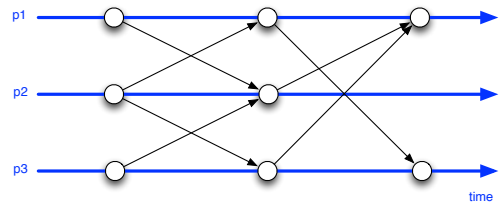


56

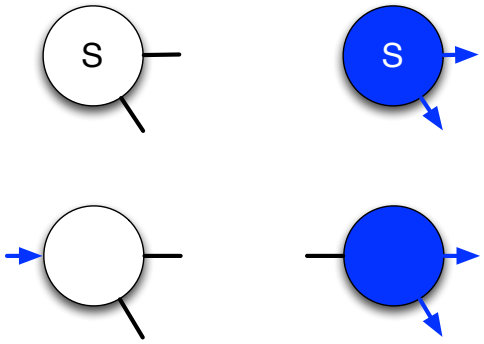
Synchronous Distributed Execution

- **Alternating sequence** of processor and link **phases**
  - In the **processor phase**, all processors that have not terminated execute their actions
  - In the **link phase**, all links execute their actions

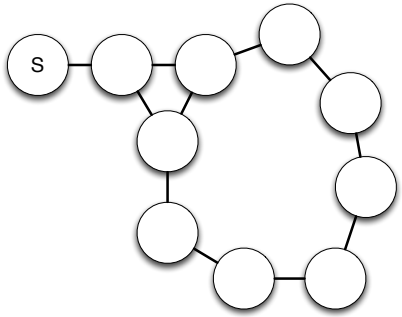
Space-Time Diagram



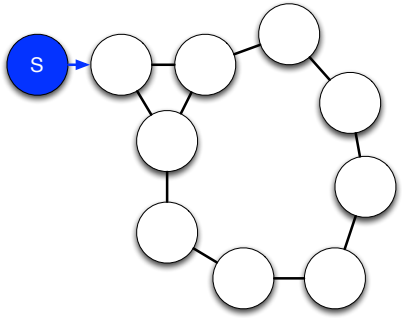
Flooding



Synchronous Flooding

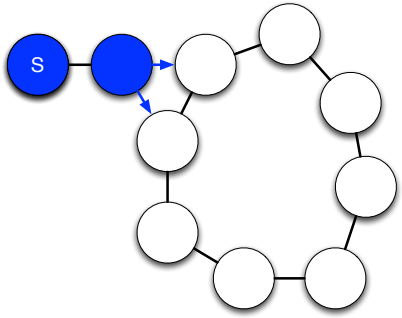


Synchronous Flooding



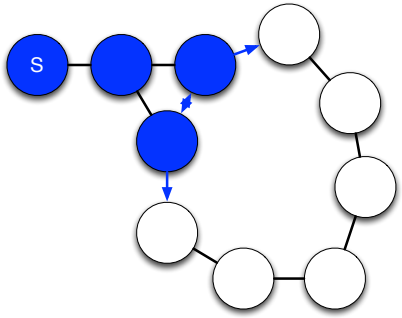
61

Synchronous Flooding



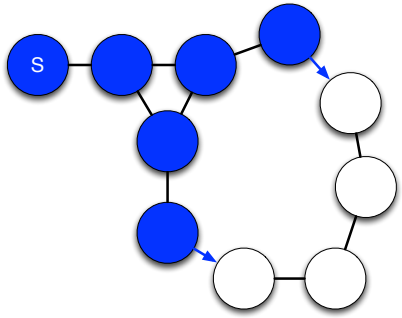
62

Synchronous Flooding



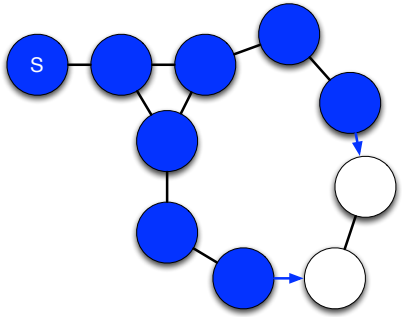
63

Synchronous Flooding



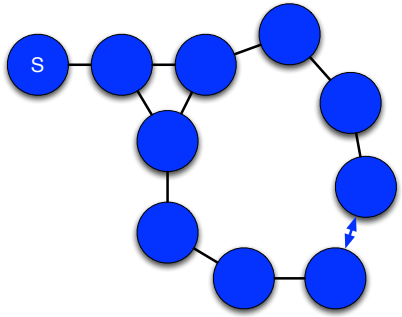
64

Synchronous Flooding



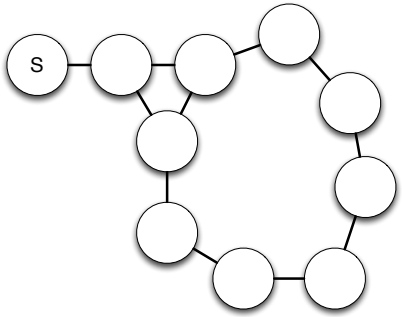
65

Synchronous Flooding



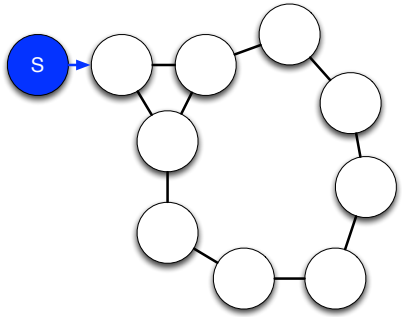
66

Asynchronous Flooding



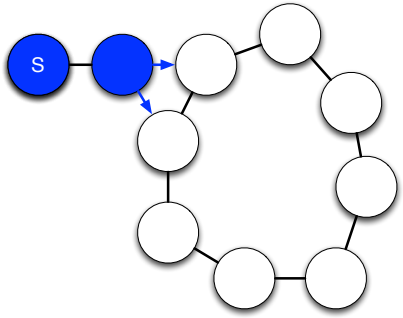
67

Asynchronous Flooding



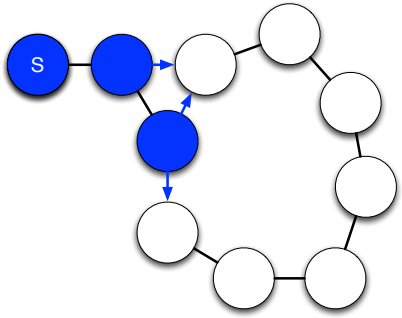
68

Asynchronous Flooding



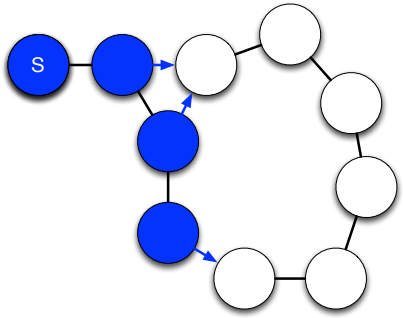
69

Asynchronous Flooding



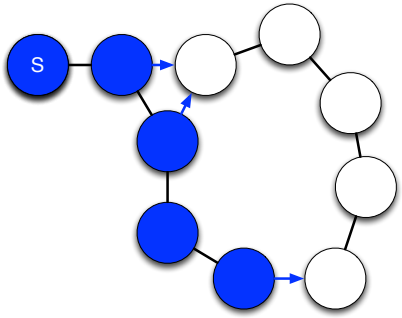
70

Asynchronous Flooding



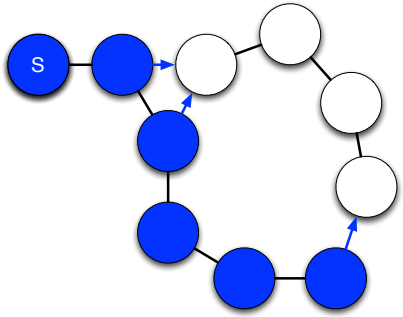
71

Asynchronous Flooding



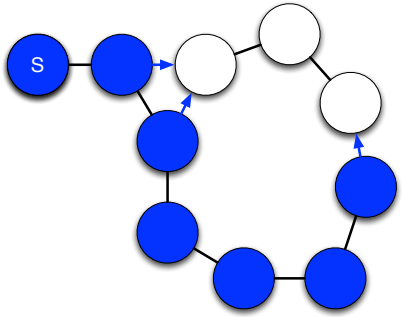
72

Asynchronous Flooding



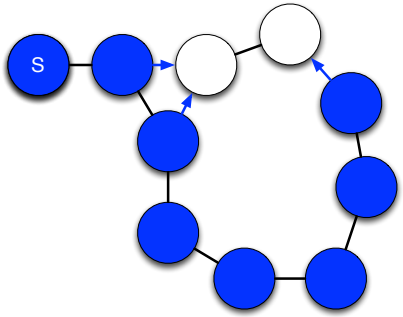
73

Asynchronous Flooding



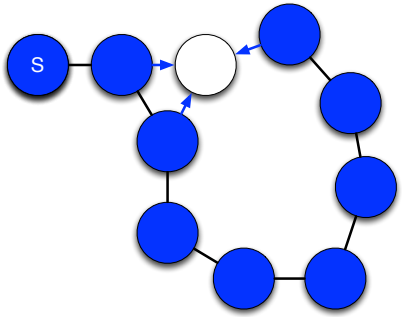
74

Asynchronous Flooding



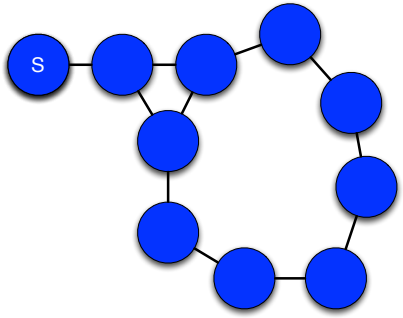
75

Asynchronous Flooding



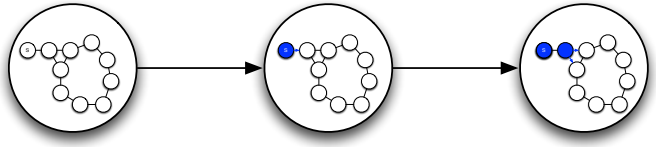
76

Asynchronous Flooding



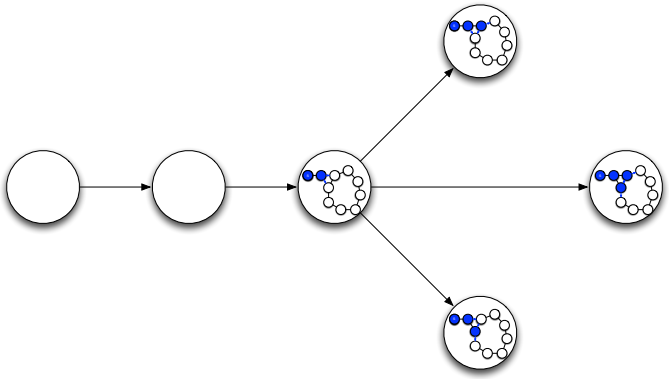
77

Configurations



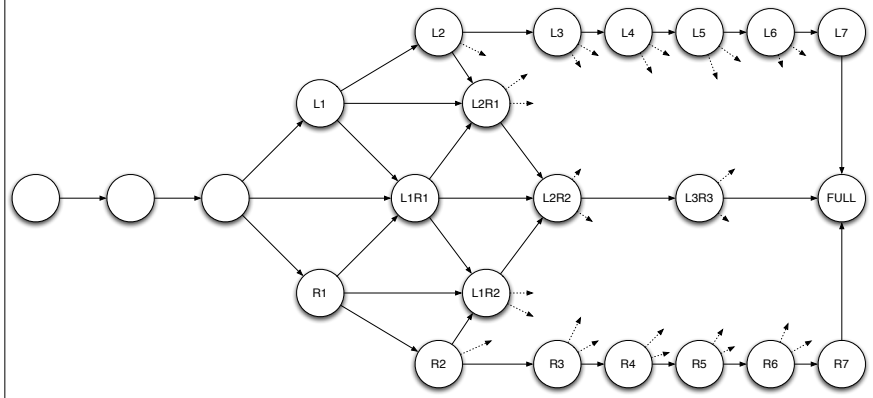
78

Configurations



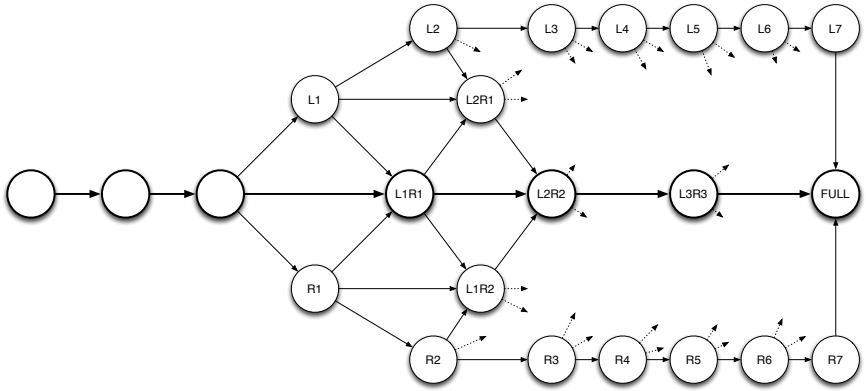
79

Configurations



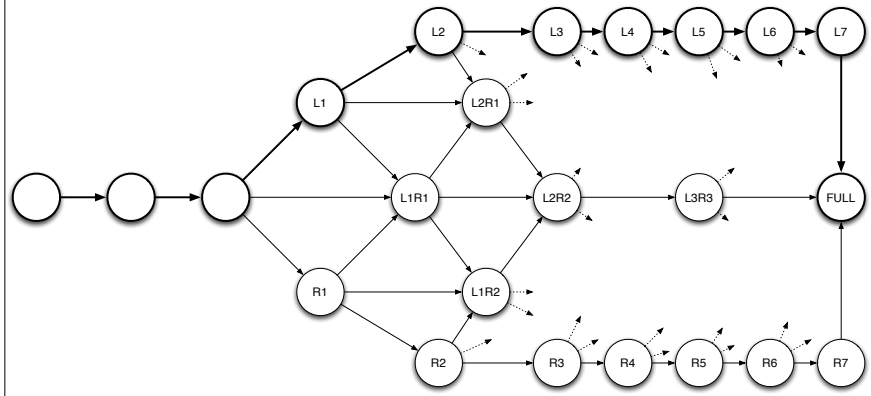
80

Executions



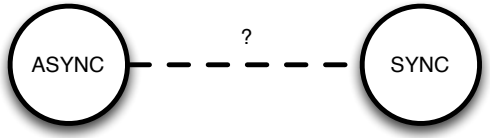
81

Executions



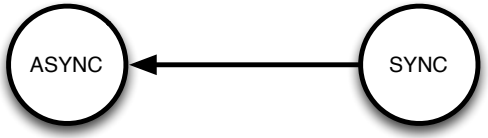
82

Synchronous *vs.* Asynchronous



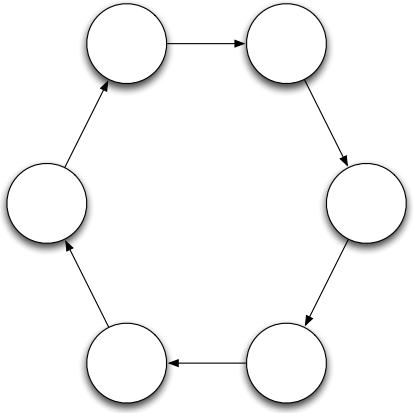
83

Synchronous *vs.* Asynchronous



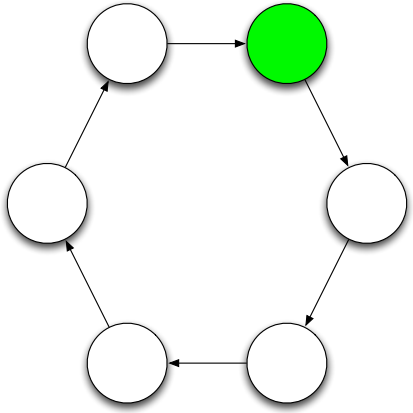
84

Leader Election



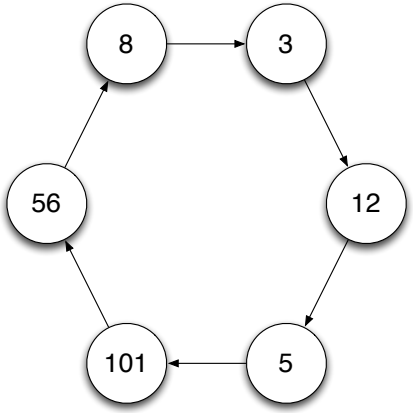
85

Leader Election



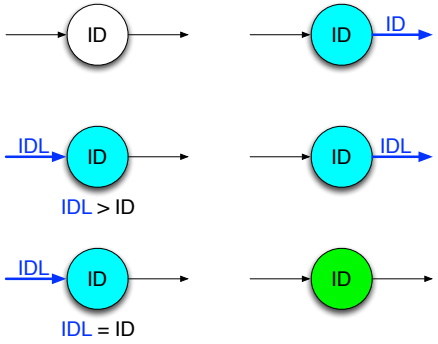
86

Leader Election



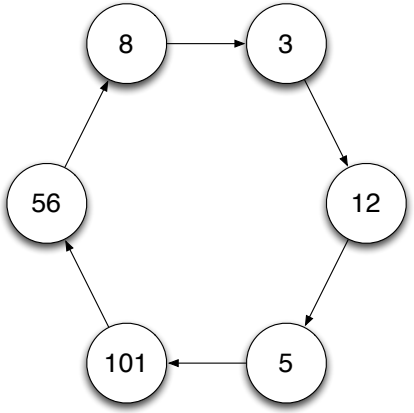
87

Leader Election



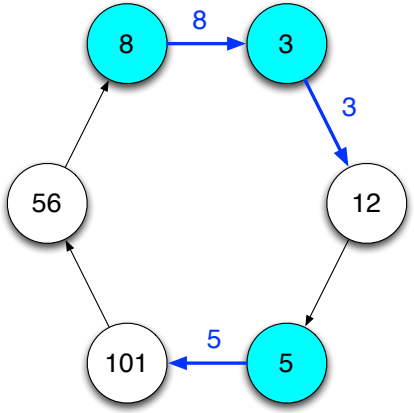
88

Leader Election



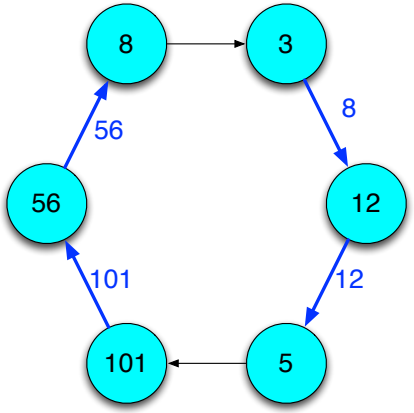
89

Leader Election



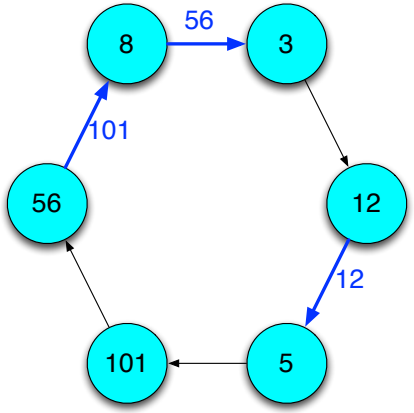
90

Leader Election



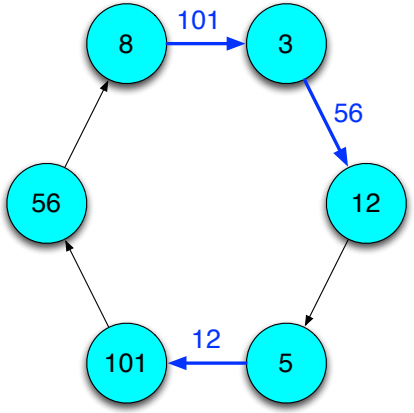
91

Leader Election



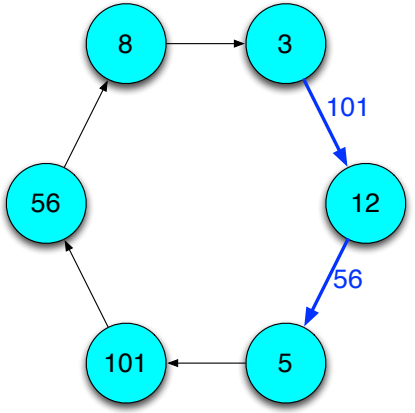
92

Leader Election



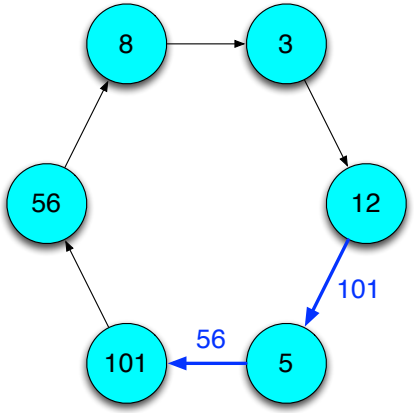
93

Leader Election



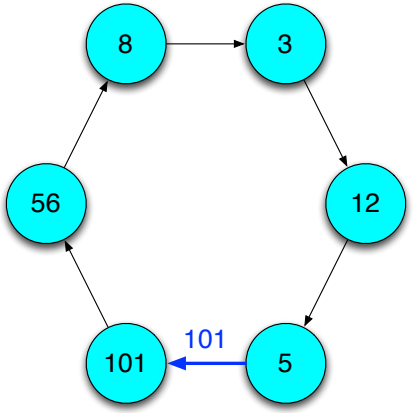
94

Leader Election



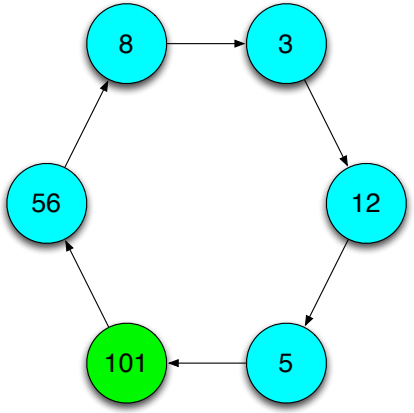
95

Leader Election



96

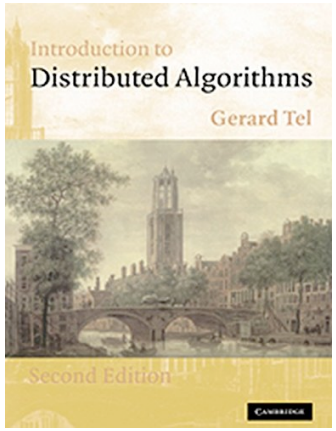
Leader Election

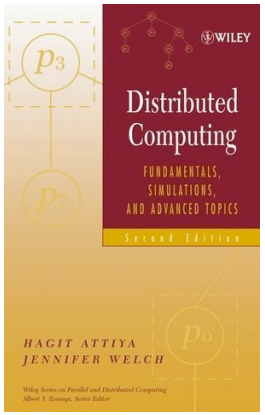


Leader Election

- **Message complexity ?**
  - Lower bound ?
- **Weaker model ?**
  - No IDs ?
  - No Orientation ?
  - General communication graph ?

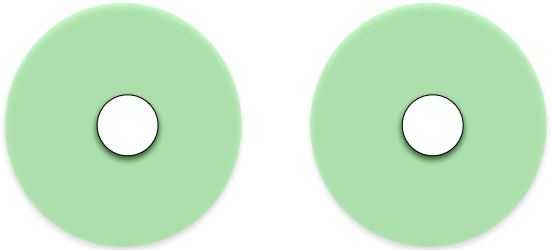
Static Networks



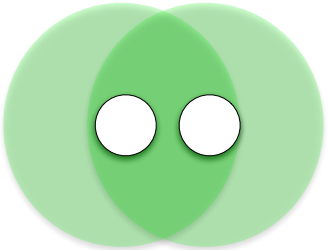


Passively Mobile Networks

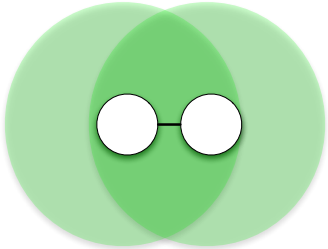
Mobility-induced Dynamic Networks



Mobility-induced Dynamic Networks



Mobility-induced Dynamic Networks

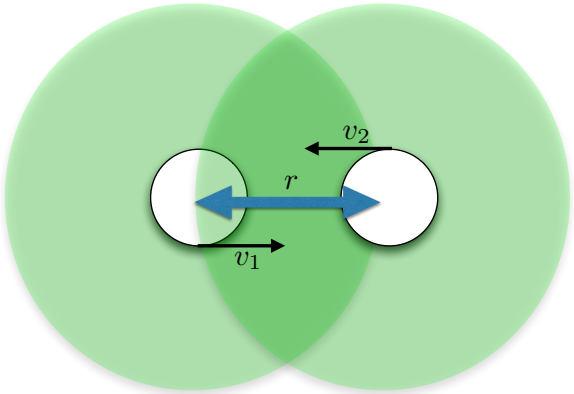


Static Algorithms for Mobile Networks

Link Lifetime

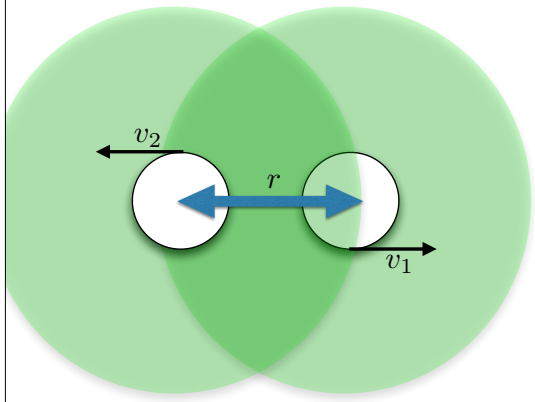
105

Link Lifetime



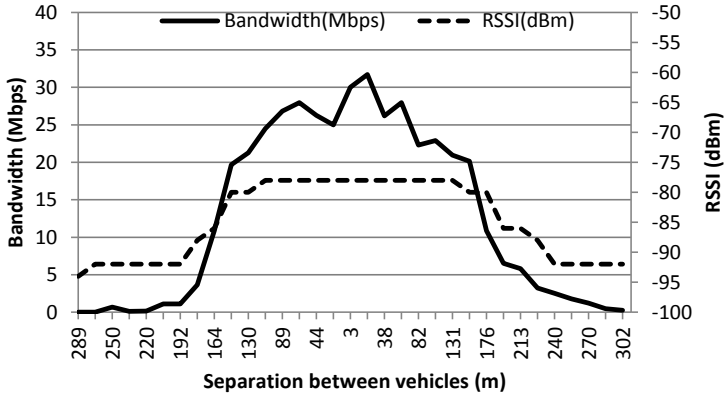
106

Link Lifetime



107

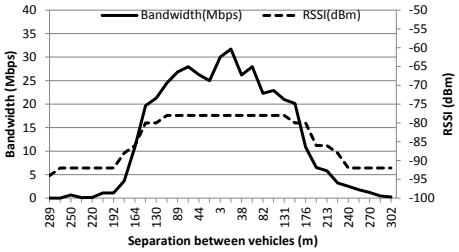
Link Lifetime



On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution. Lerotholi S. Mojela ; Marthinus J. Booysen, Industrial Technology (ICIT), 2013 IEEE International Conference on

108

Link Lifetime



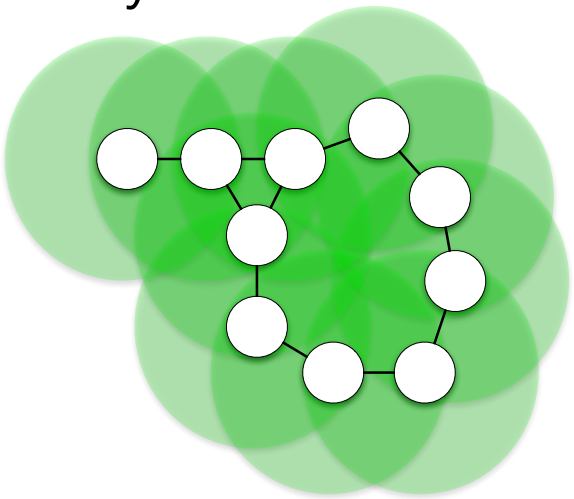
On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution. Lerotholi S. Mojela ; Marthinus J. Booysen, Industrial Technology (ICIT), 2013 IEEE International Conference on

Fig. 4b shows a graph of the two vehicles traveling in opposite directions at an average relative speed of 64 km/h in an urban area. The average contact time recorded was 33s and the average maximum communication range was found to be 302 m with an average bandwidth of 13.7 Mbps per test run taken over the period of established contact, average jitter of 1.88 ms and an average of 51.7 MB data transferred per contact period. The maximum peak bandwidth of 31.7 Mbps

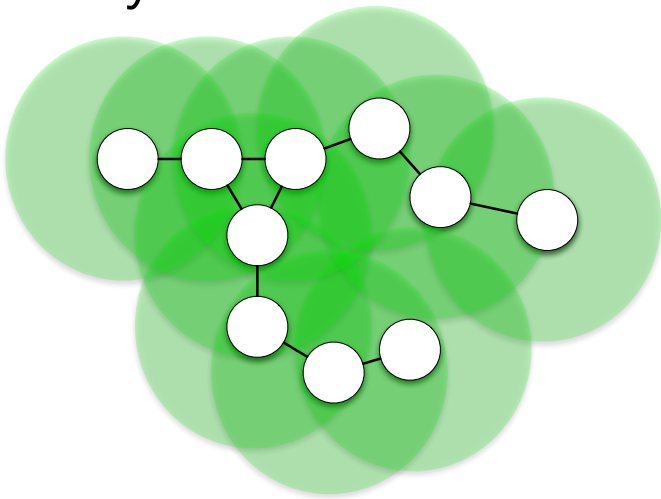
Round-trip Time

	Barcelona	Paris	Tokyo	Toronto	Washington
Amsterdam	44.173ms	11.876ms	242.819ms	96.619ms	93.342ms
Auckland	295.323ms	280.111ms	241.541ms	199.854ms	211.48ms
Berlin	49.71ms	20.041ms	275.013ms	103.027ms	113.909ms
Copenhagen	43.745ms	25.574ms	272.459ms	117.751ms	100.623ms
Dallas	143.543ms	114.941ms	145.957ms	76.173ms	34.194ms
London	35.431ms	4.542ms	240.519ms	85.102ms	75.484ms
Los Angeles	157.289ms	141.484ms	112.265ms	58.689ms	71.881ms
Moscow	74.806ms	55.67ms	299.067ms	137.808ms	144.68ms
New York	130.6ms	77.551ms	171.641ms	17.78ms	47.161ms
Paris	31.085ms	—	260.569ms	103.015ms	107.643ms
Stockholm	59.785ms	27.894ms	285.462ms	123.358ms	115.657ms
Tokyo	267.753ms	268.811ms	—	172.841ms	186.818ms

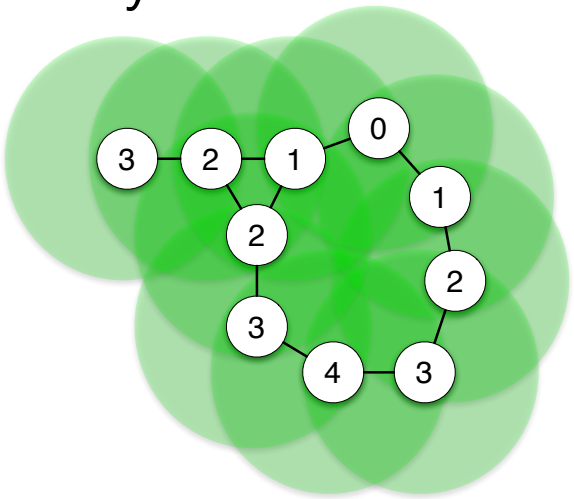
Mobility *vs.* Global State



Mobility *vs.* Global State

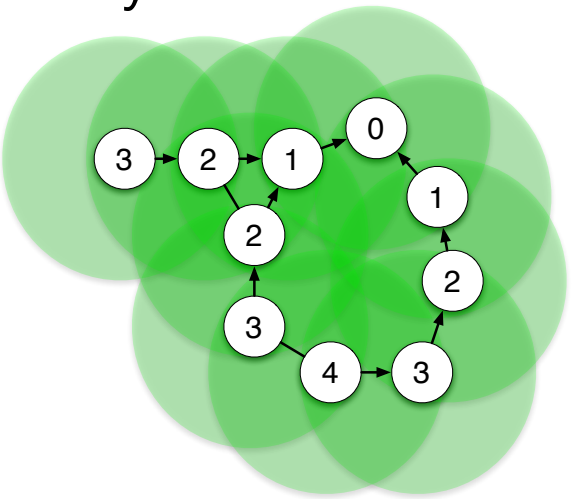


Mobility *vs.* Global State



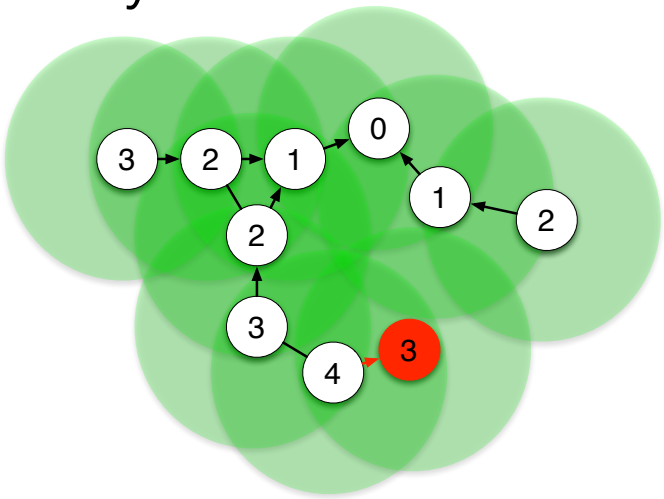
113

Mobility *vs.* Global State



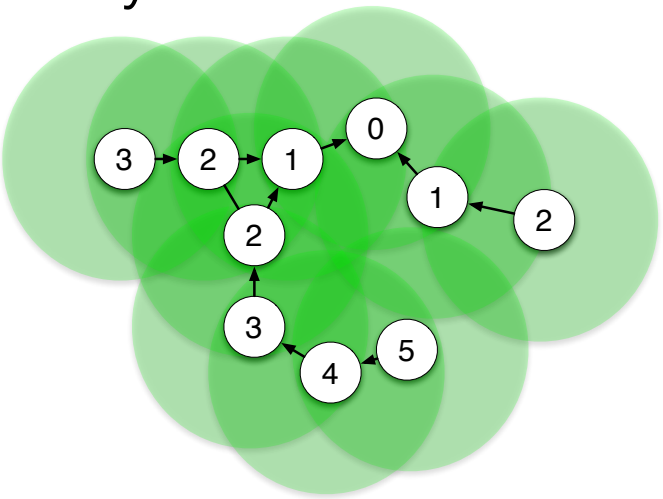
114

Mobility *vs.* Global State



115

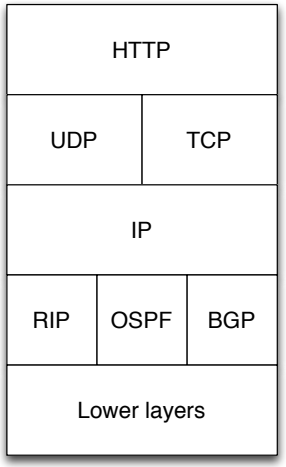
Mobility *vs.* Global State



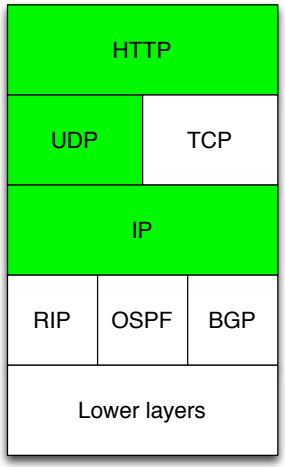
116

Stateless Algorithms

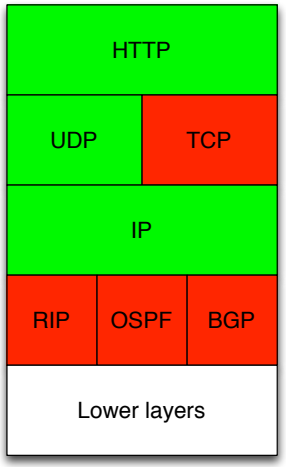
Statelessness



Statelessness



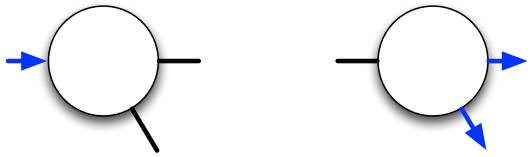
Statelessness



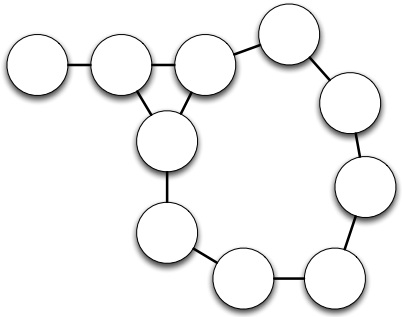
Stateless Routing

A routing algorithm is **stateless** if it is designed such that devices store *no information* about messages *between transmissions*. It is **stateful** otherwise.

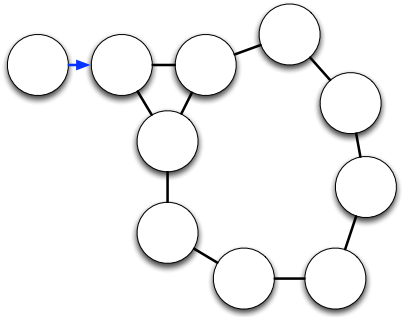
Flooding



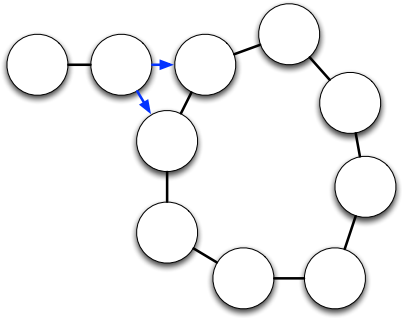
Stateless Flooding



Stateless Flooding

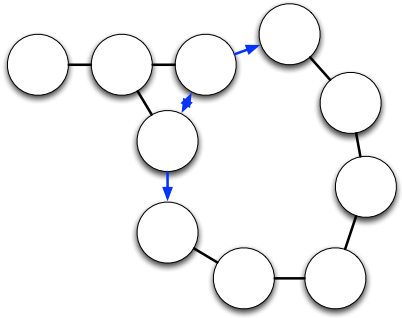


Stateless Flooding



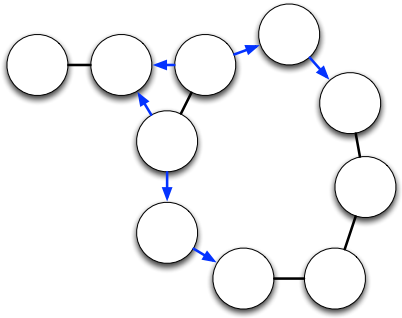
125

Stateless Flooding



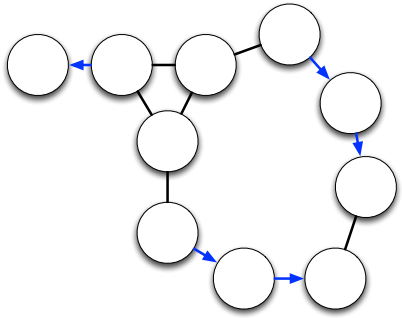
126

Stateless Flooding



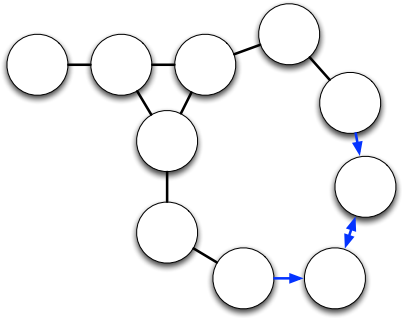
127

Stateless Flooding



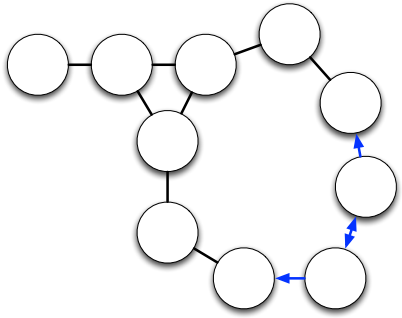
128

Stateless Flooding



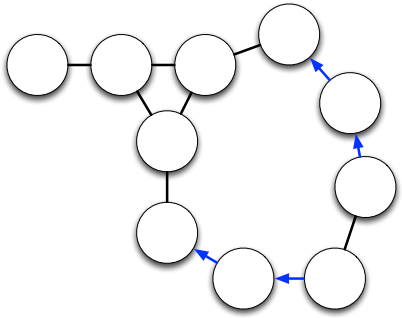
129

Stateless Flooding



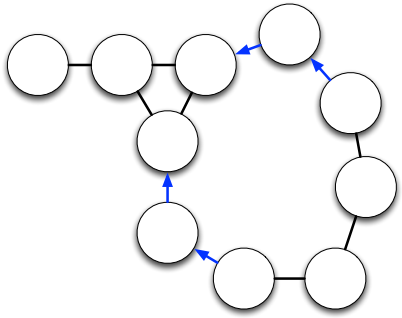
130

Stateless Flooding



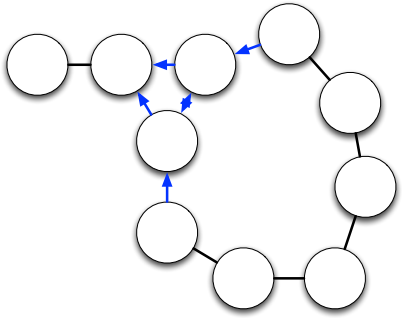
131

Stateless Flooding



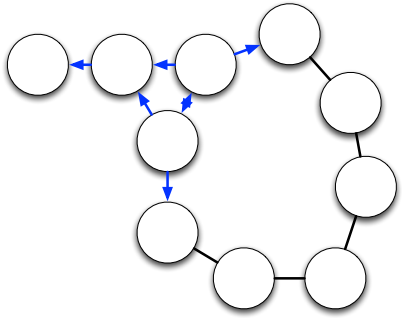
132

Stateless Flooding



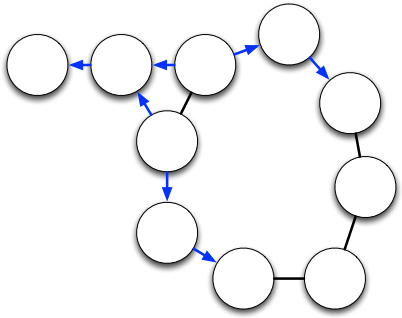
133

Stateless Flooding



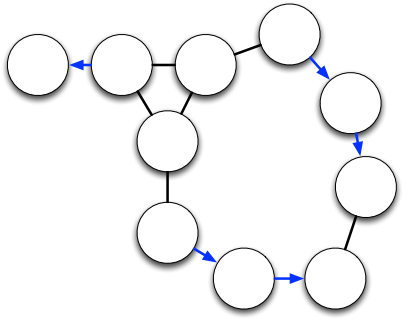
134

Stateless Flooding



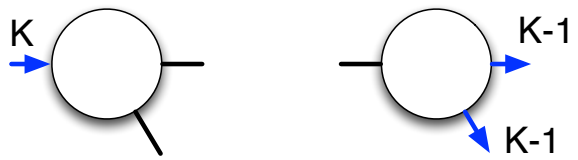
135

Stateless Flooding

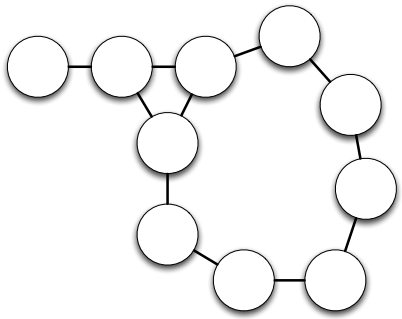


136

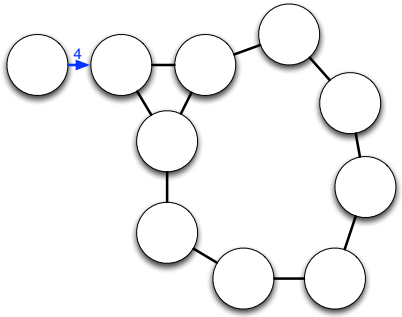
Flooding v2



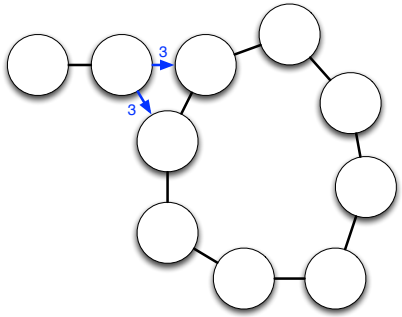
TTL Flooding



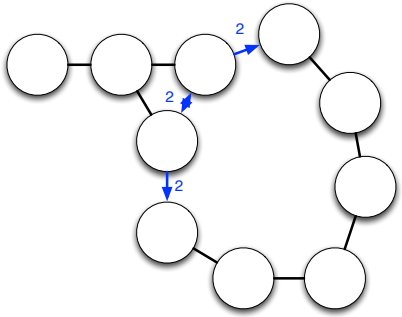
TTL Flooding



TTL Flooding

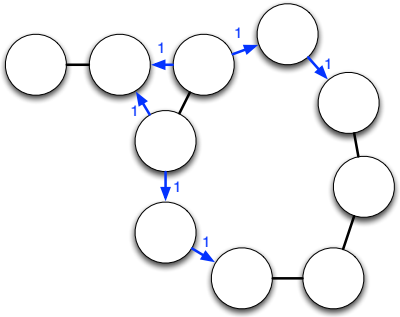


TTL Flooding



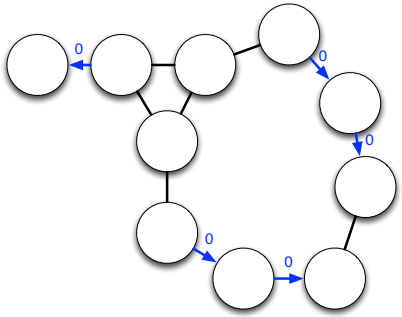
141

TTL Flooding



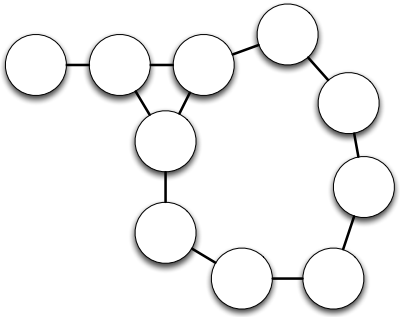
142

TTL Flooding



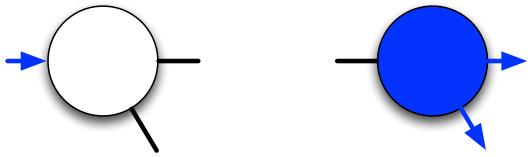
143

TTL Flooding



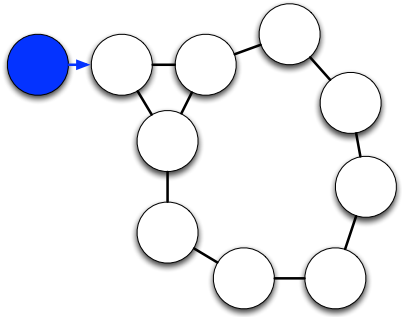
144

Flooding v3



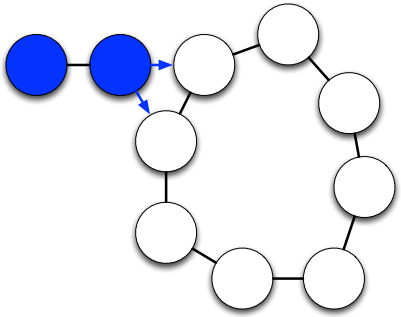
145

Stateful Flooding



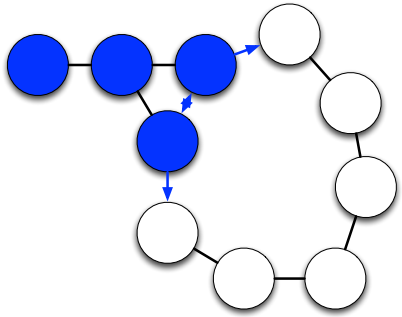
146

Stateful Flooding



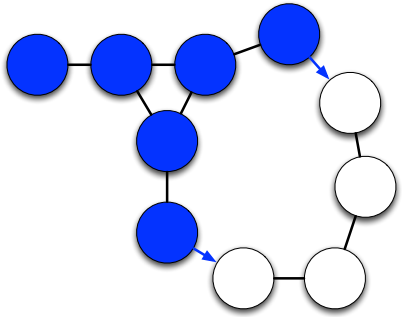
147

Stateful Flooding



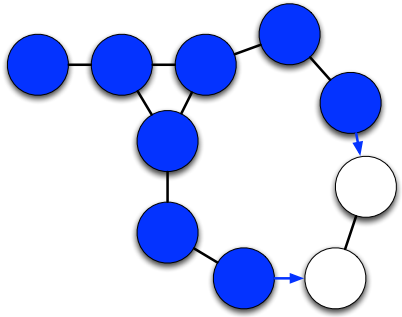
148

Stateful Flooding



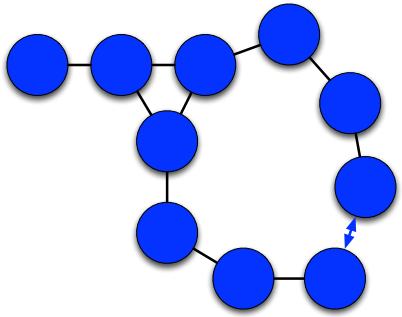
149

Stateful Flooding



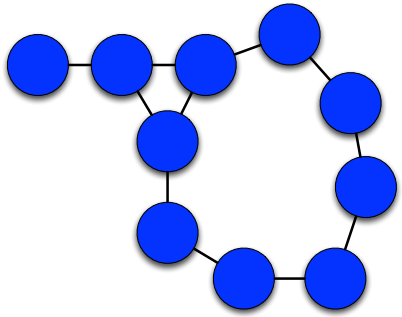
150

Stateful Flooding



151

Stateful Flooding



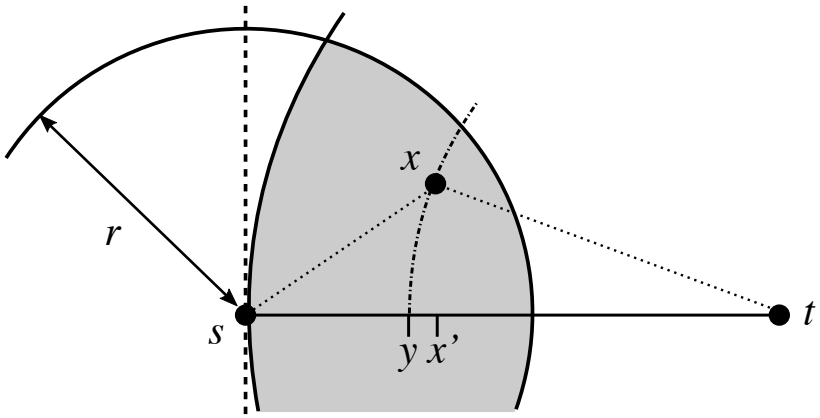
152

Geometric Routing

- Each node is aware of its *coordinates* (and those of its neighbors)
- The message contains the coordinates of the destination
- **Goal**: deliver the message to the destination *without routing tables*

Stojmenovic, Ivan (2002). "Position based routing in ad hoc networks". IEEE Communications Magazine. 40 (7): 128–134.

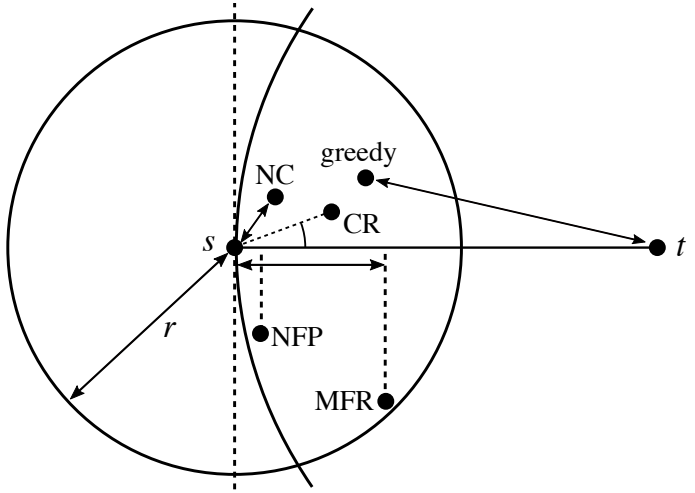
Progress *vs.* Distance



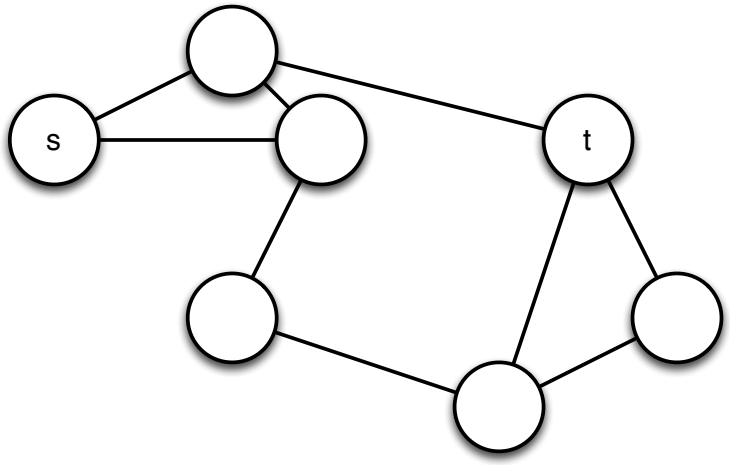
Which Criterion?

- **MFR**: most forwarding progress
- **CR**: minimize angular criterion
- **Greedy**: minimize distance to destination
- **NC**: nearest closer
- **NFP**: nearest with forwarding progress

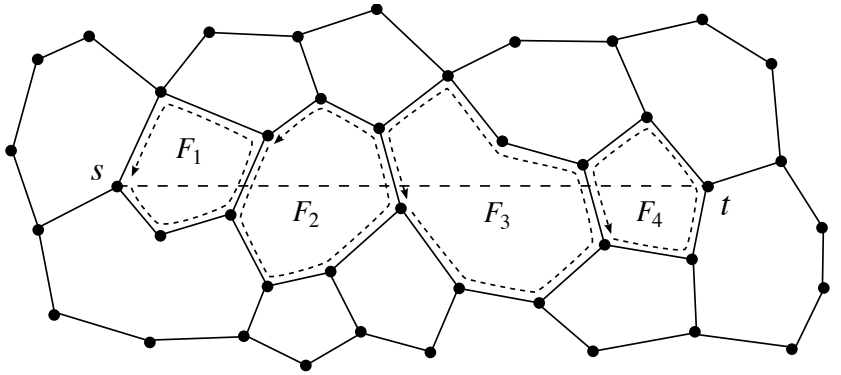
Which Criterion?



Delivery Guarantee?

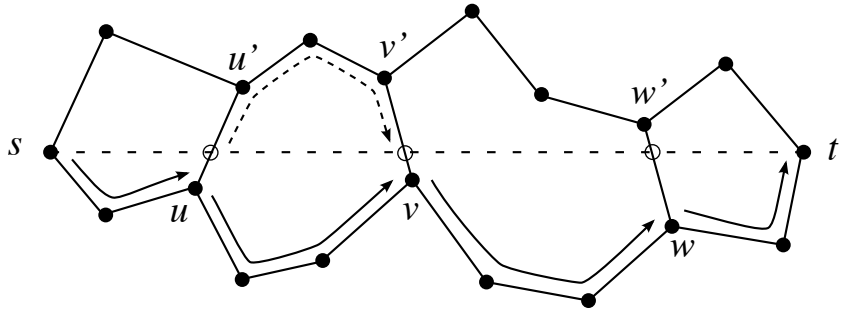


Planar Graph Routing

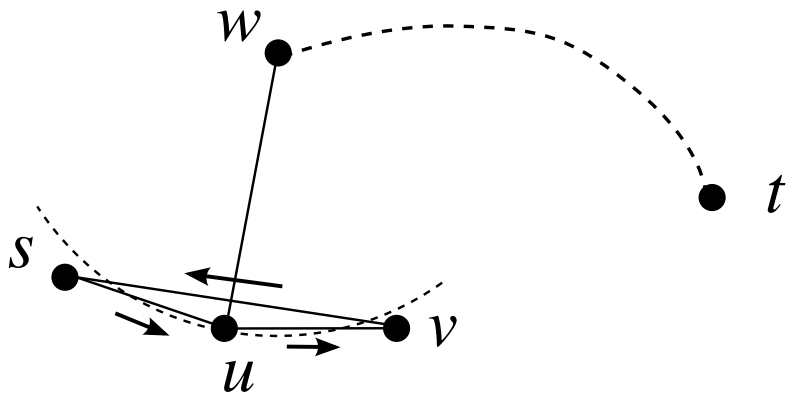


Bose, P.; Morin, P.; Stojmenovic, I.; Urrutia, J. (1999). "Routing with guaranteed delivery in ad hoc wireless networks". Proc. of the 3rd international workshop on discrete algorithms and methods for mobile computing and communications (DIALM '99). pp. 48–55.

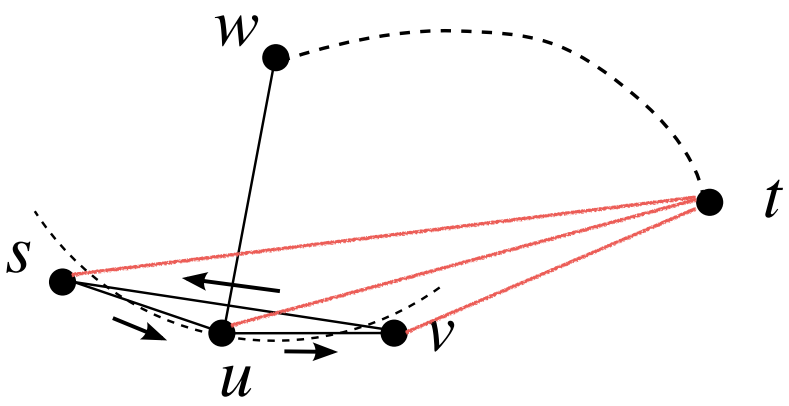
Face Routing



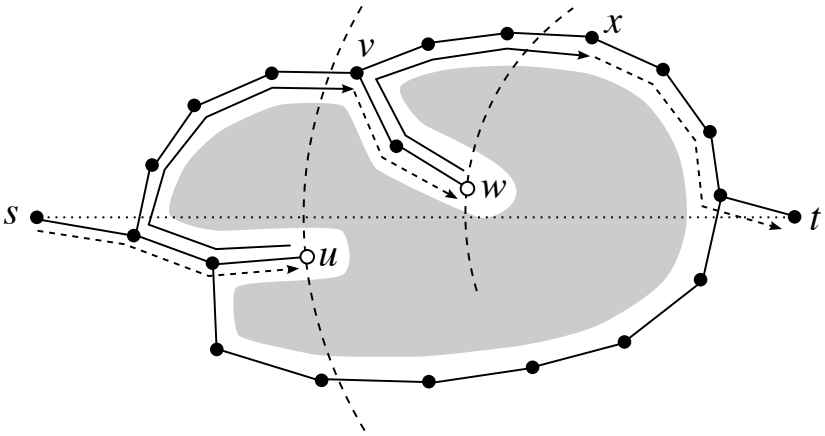
Planar Graphs!



Planar Graphs!



Greedy / Face / Greedy



Self-stabilization

Self-stabilization 101

Example

$U_0 = a$

$U_{n+1} = \frac{U_n}{2}$ if U_n is even

$U_{n+1} = \frac{3U_n+1}{2}$ if U_n is odd

Example

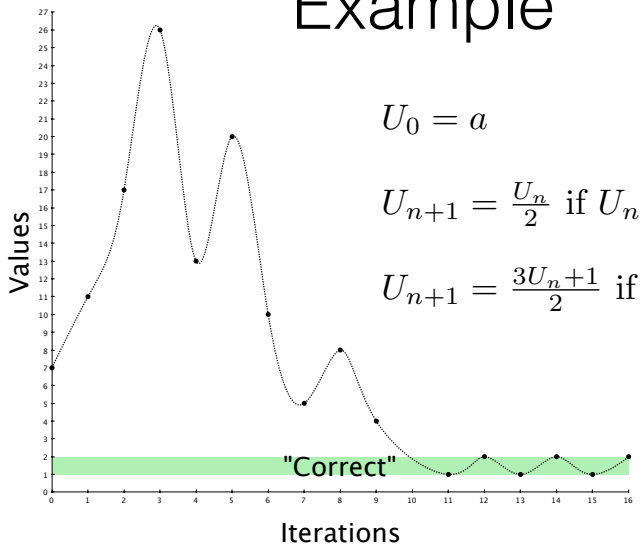
$U_0 = a$

$U_{n+1} = \frac{U_n}{2}$ if U_n is even

$U_{n+1} = \frac{3U_n+1}{2}$ if U_n is odd

n	0	1	2	3	4	5	6	7	8	9	10	11	12
U_n	7	11	17	26	13	20	10	5	8	4	2	1	2

Example

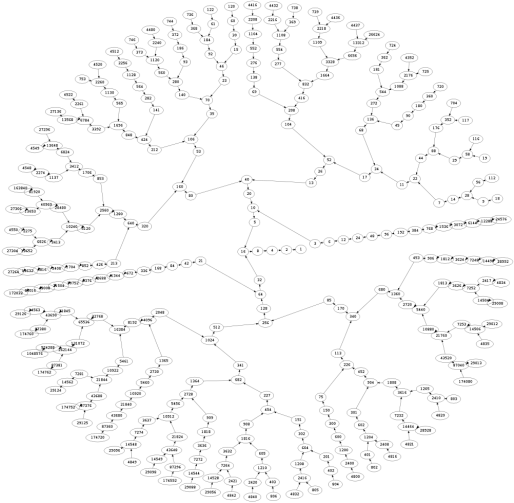


$U_0 = a$

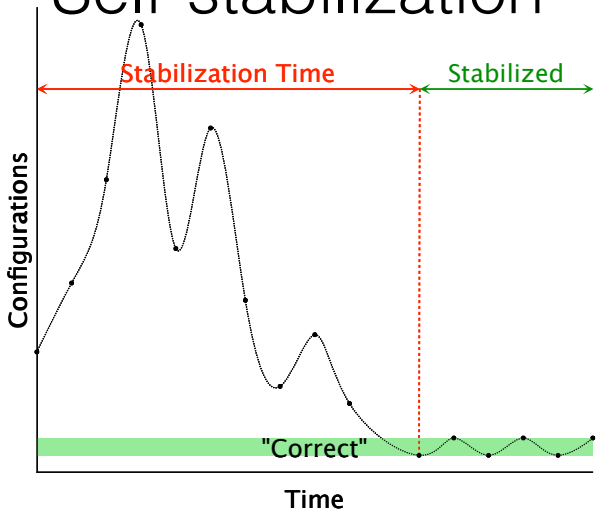
$U_{n+1} = \frac{U_n}{2}$ if U_n is even

$U_{n+1} = \frac{3U_n+1}{2}$ if U_n is odd

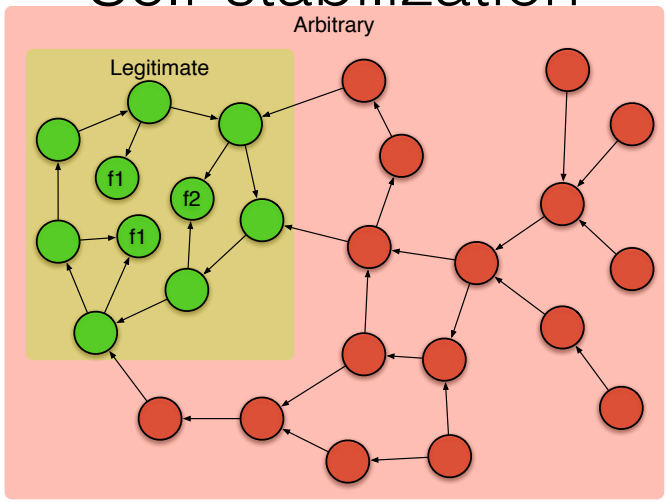
Example



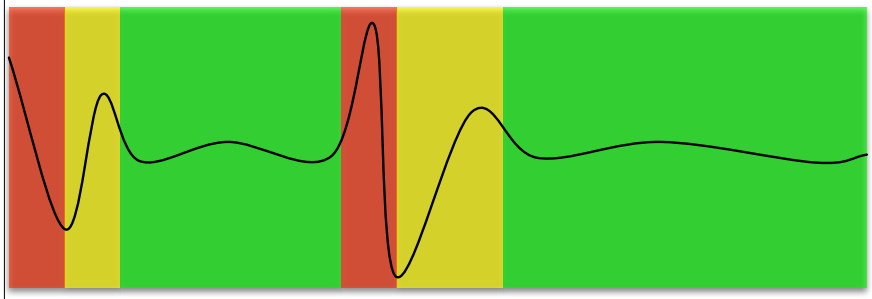
Self-stabilization



Self-stabilization



Self-stabilization



Distributed Systems

- **Configuration**: product of the local states of system components
- **Execution**: interleaving of the local executions of the system components

Distributed Systems

- **Classical**: Starting from a particular initial configuration, the system immediately exhibits correct behavior
- **Self-stabilizing**: Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct

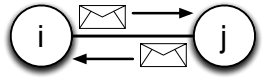
Distributed Systems

- **Self-stabilizing**: Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct
  - Defined by *Dijkstra* in 1974
  - Advocated by *Lamport* in 1984 to address fault-tolerant issues
  - Stale states due to **mobility** can be recovered!

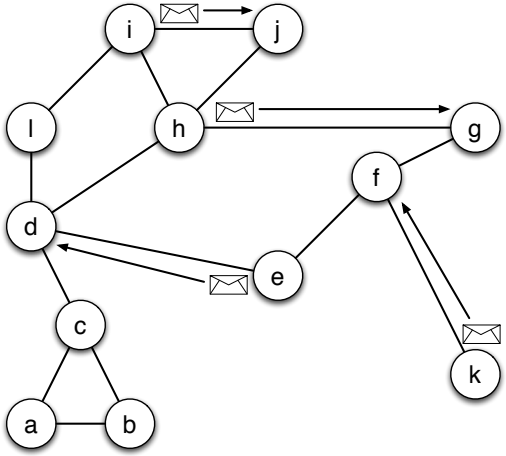
Configurations

```
int x = 0;
...
if( x == 0 ) {
  // code assuming x equals 0
}
else {
  // code assuming x does not equal 0
}
```


Configurations



Configurations



Hypotheses

Atomicity

- A «*stabilizing*» sequential program

```
int x = 0;
...
while( x == x ) {
  x = 0;
  // code assuming x equals 0
}
```

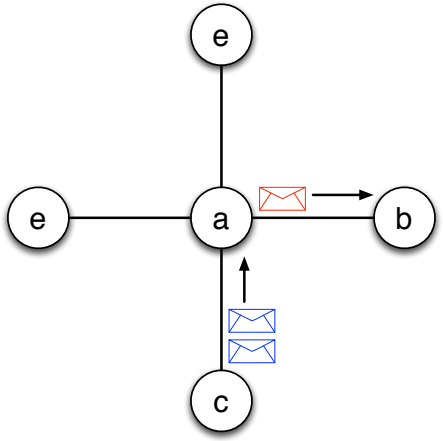

Atomicity

- A «*stabilizing*» sequential program

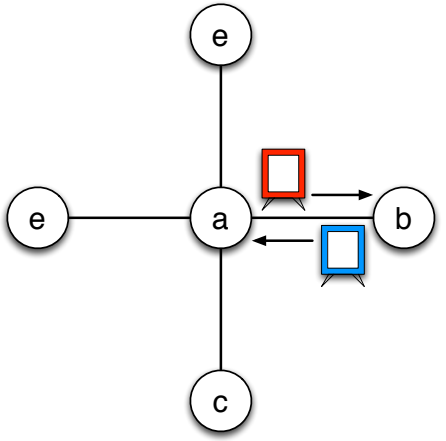
```
0 iconst_0
1 istore_1
2 goto 7
5 iconst_0
6 istore_1
7 iload_1
8 iload_1
9 if_icmpeq 5
```

Problem

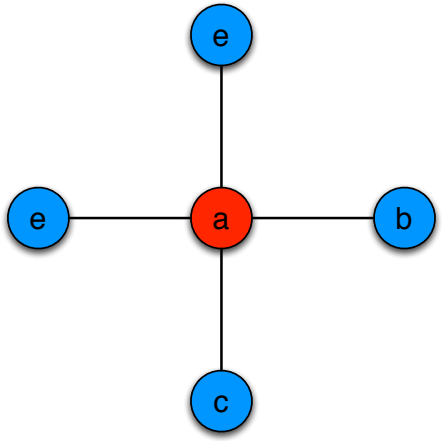
Communications



Communications

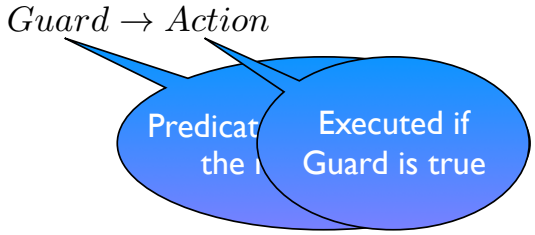


Communications



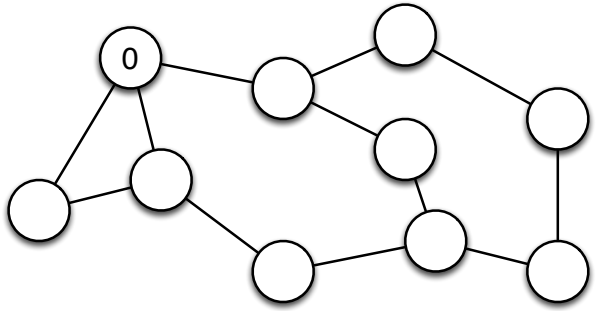
Example

- **Shared memory**: in one atomic step, read the state of all neighbors and write own state
- **Guarded command**



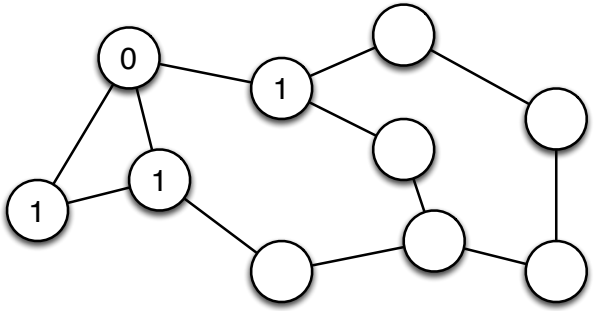
Example

$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



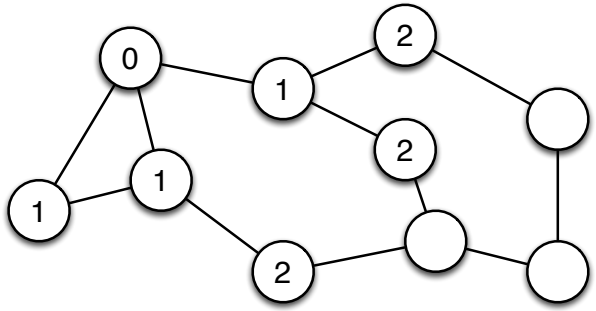
Example

$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



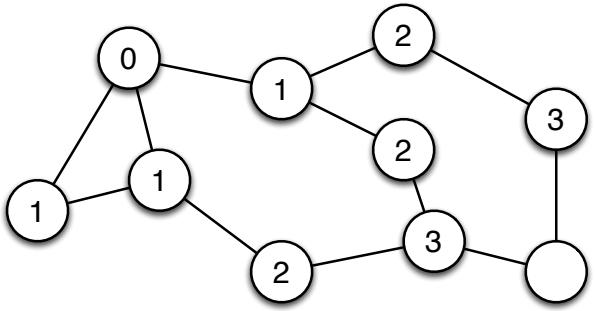
Example

$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



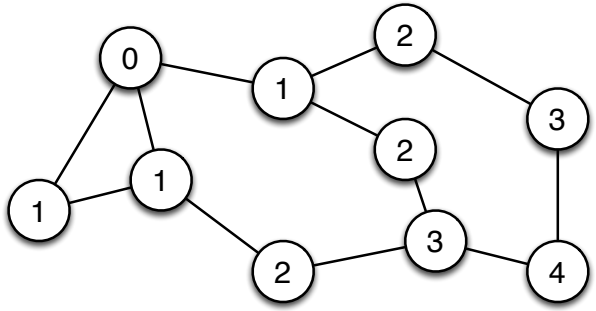
Example

$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



Example

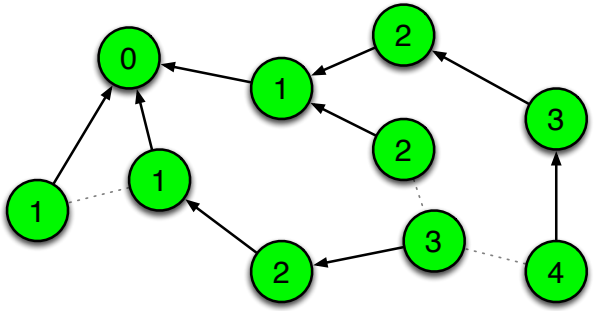
$$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$$



189

Example

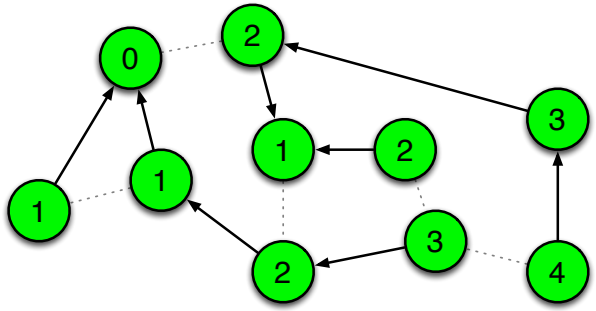
$$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$$



190

Example

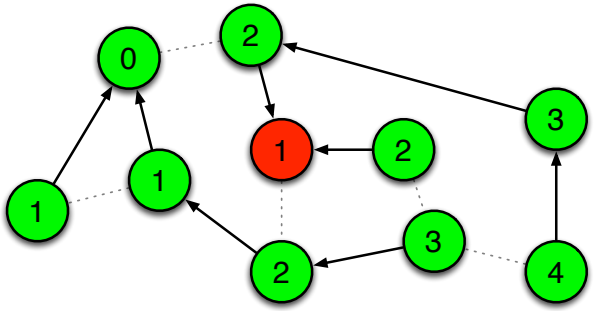
$$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$$



191

Example

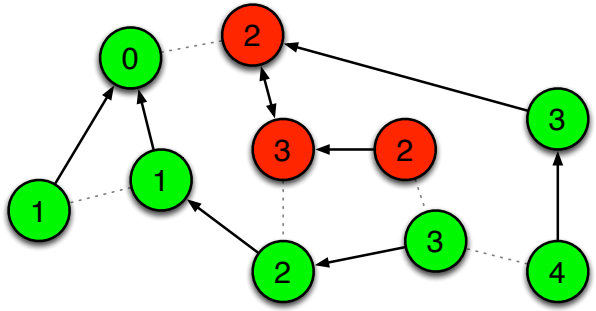
$$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$$



192

Example

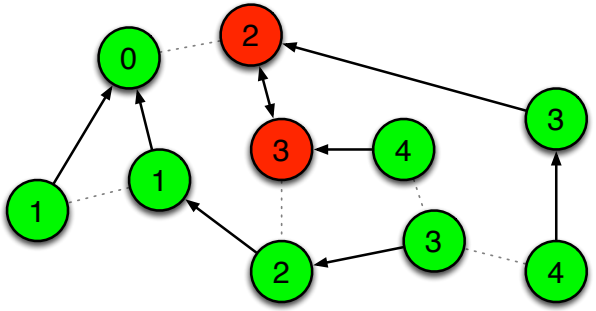
$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



193

Example

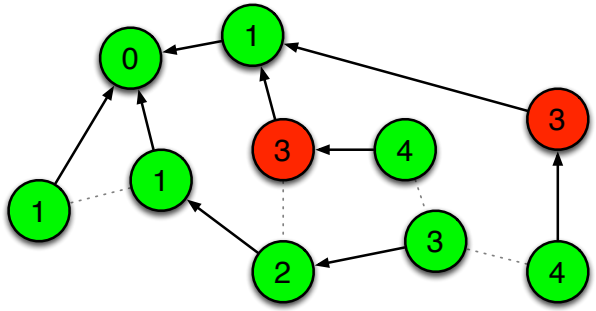
$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



194

Example

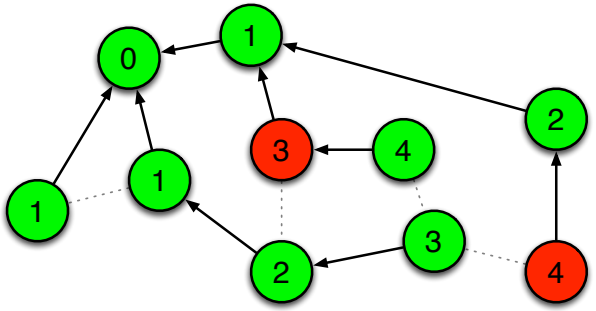
$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



195

Example

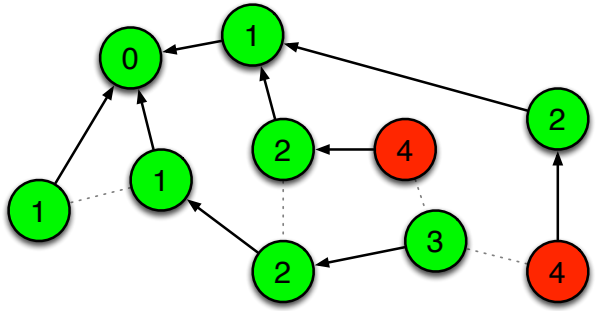
$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$



196

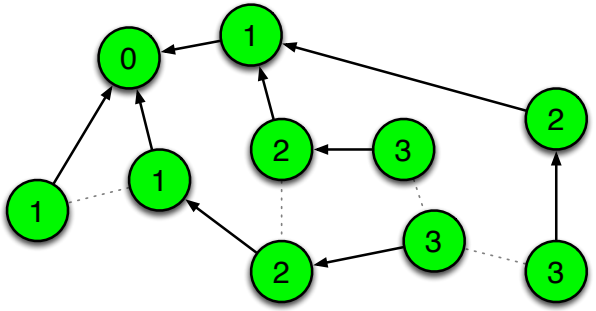
Example

$$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$$



Example

$$true \rightarrow Distance_i := Min_{j \in Neighbors_i}\{Distance_j + 1\}$$



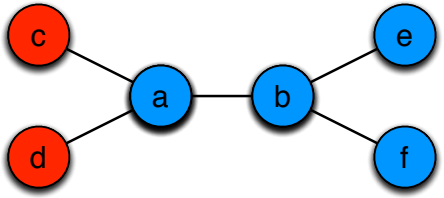
Scheduling

- **Scheduler** (a.k.a. **Daemon**): the daemon chooses among activatable processes those that will execute their actions
  - can be seen as an *adversary* whose role is to prevent stabilization

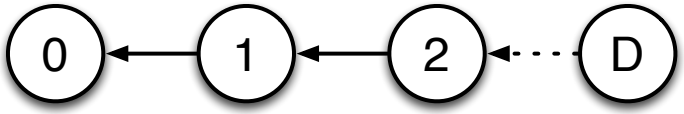
Spatial Scheduling

$$true \rightarrow color_i := Min\{\Delta \setminus \{color_j | j \in Neighbors_i\}\}$$

$$\Delta = \{\bigcirc\bigcirc\bigcirc\bigcirc\}$$



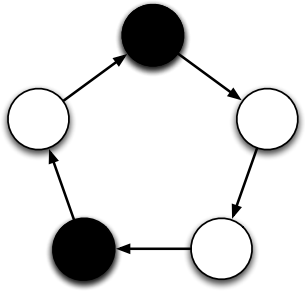
Spatial Scheduling



Temporal Scheduling

$token \rightarrow pass\ token\ to\ left\ neighbor\ with\ probability\ \frac{1}{2}$

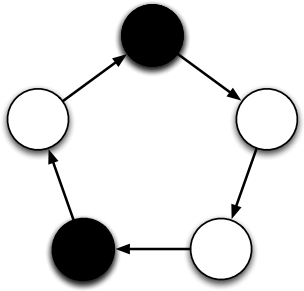
$token =$ ● $\quad no\ token =$ ○



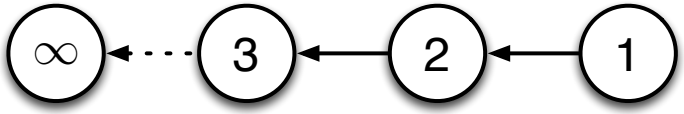
Temporal Scheduling

$token \rightarrow pass\ token\ to\ left\ neighbor\ with\ probability\ \frac{1}{2}$

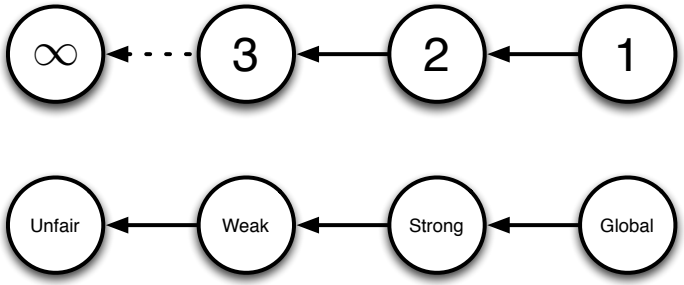
$token =$ ● $\quad no\ token =$ ○



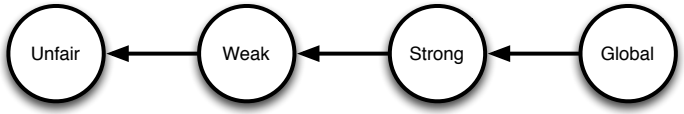
Temporal Scheduling



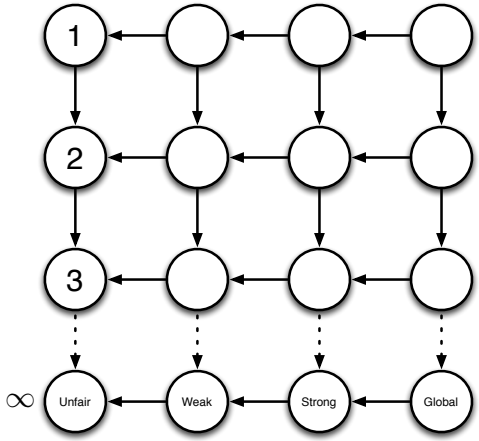
Temporal Scheduling



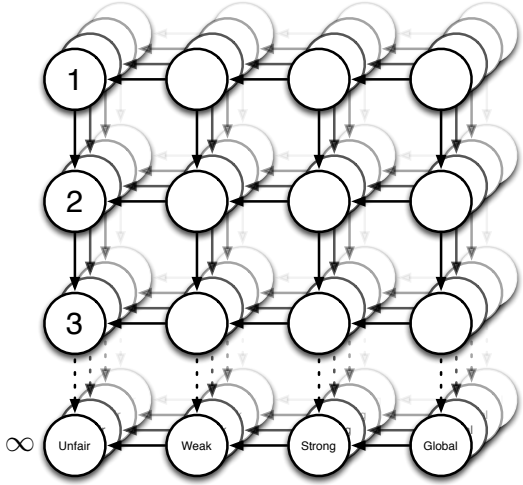
A Map of Daemons



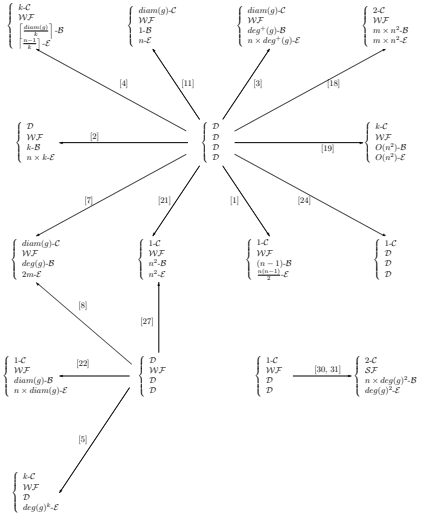
A Map of Daemons



A Map of Daemons

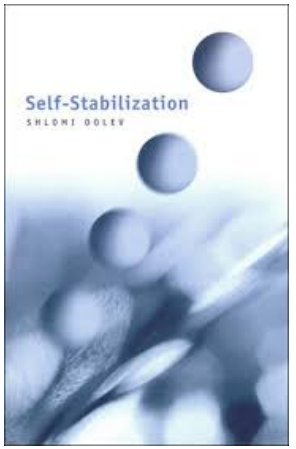


Daemon Transformers



209

Self-stabilization



210

Population Protocols

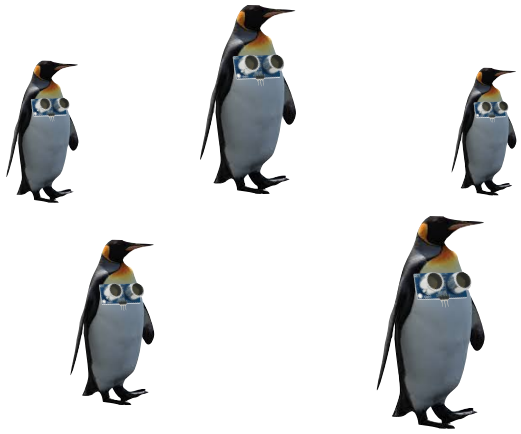
211

Population Protocols

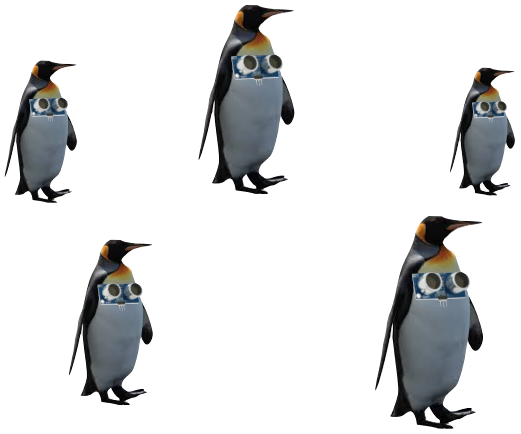
Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, René Peralta: Computation in networks of passively mobile finite-state sensors. Distributed Computing 18(4): 235-253 (2006)

212

Population Protocols



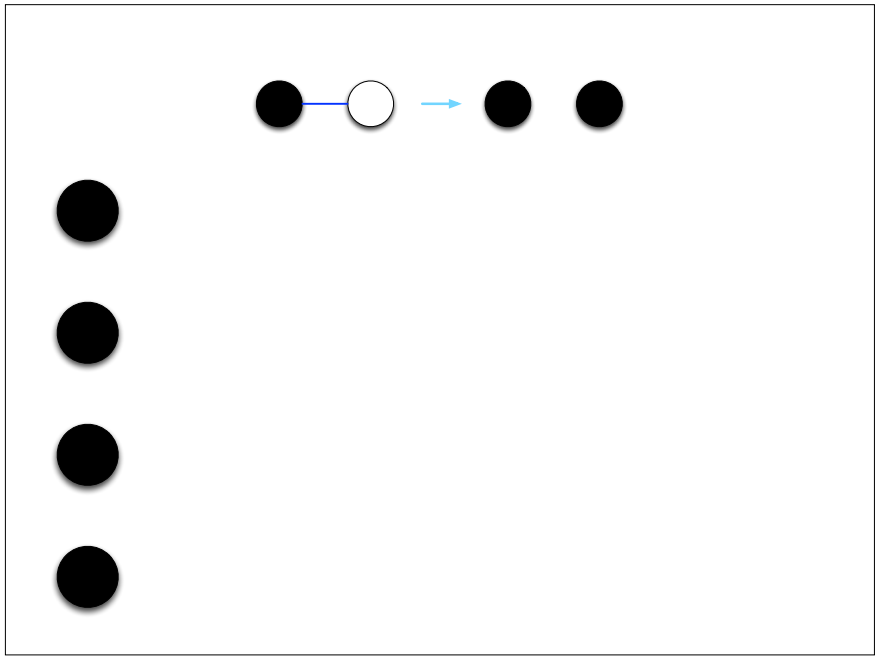
Population Protocols



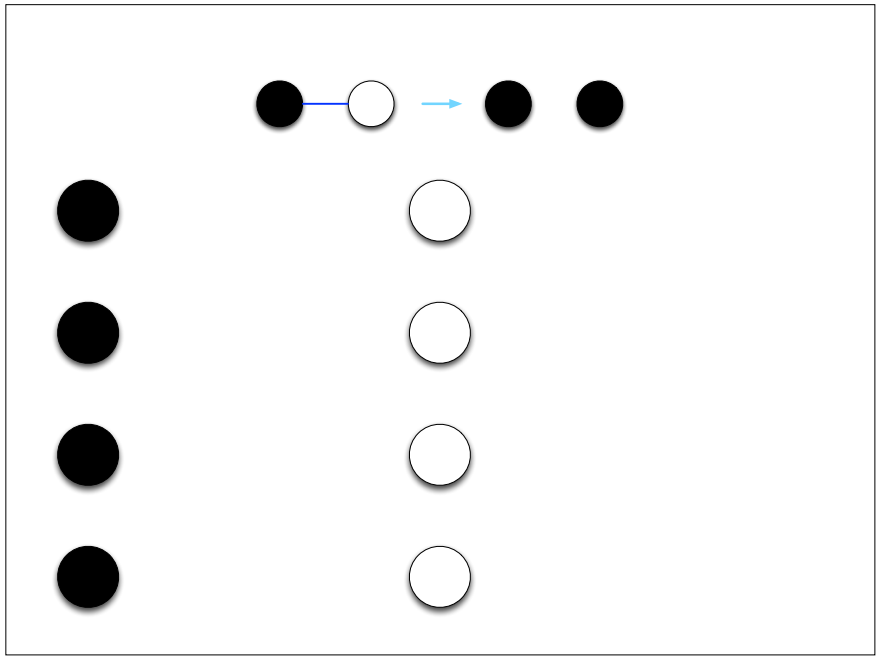
Population Protocols

- **Definition**
  - A *Population Protocol* is a 6-tuple **(X,Y,Q,I,O,T)**
    - **X**: Set of inputs
    - **Y**: Set of outputs
    - **Q**: Set of states
    - **I**: Input mapping function, $X \longrightarrow Q$
    - **O**: Output mapping function, $Q \longrightarrow O$
    - **T**: Transition function, $Q \times Q \longrightarrow Q \times Q$

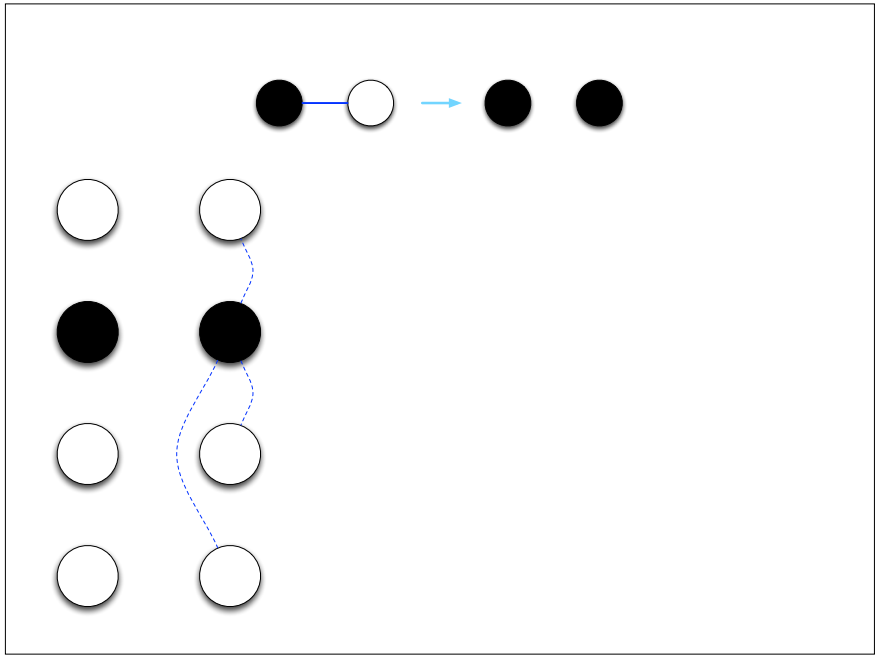
Example 1



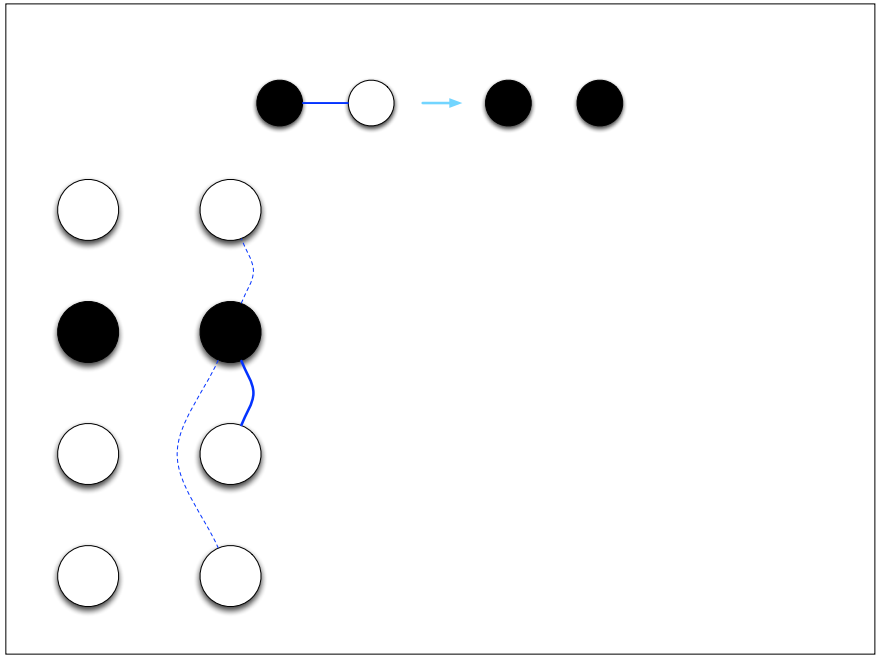
217



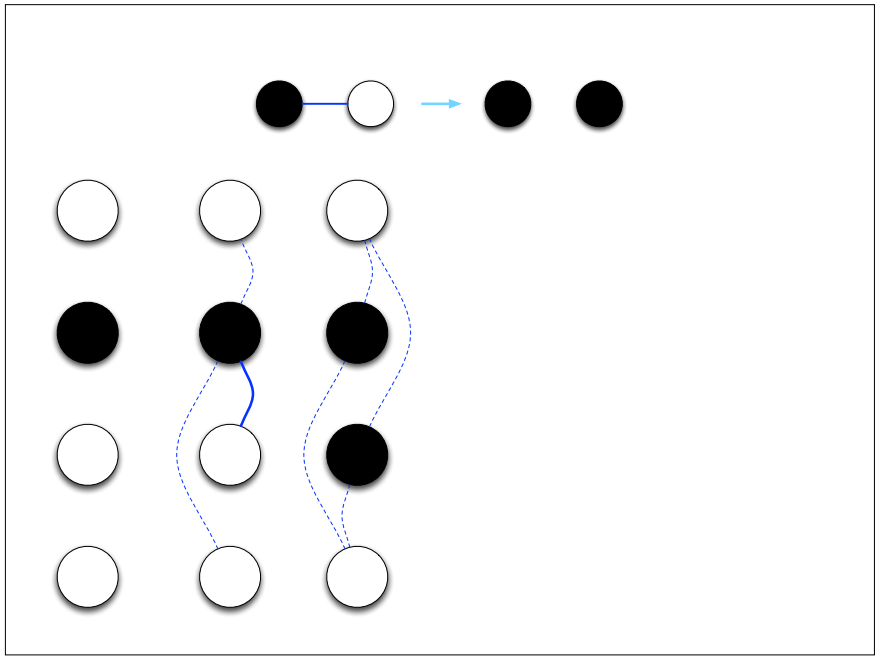
218



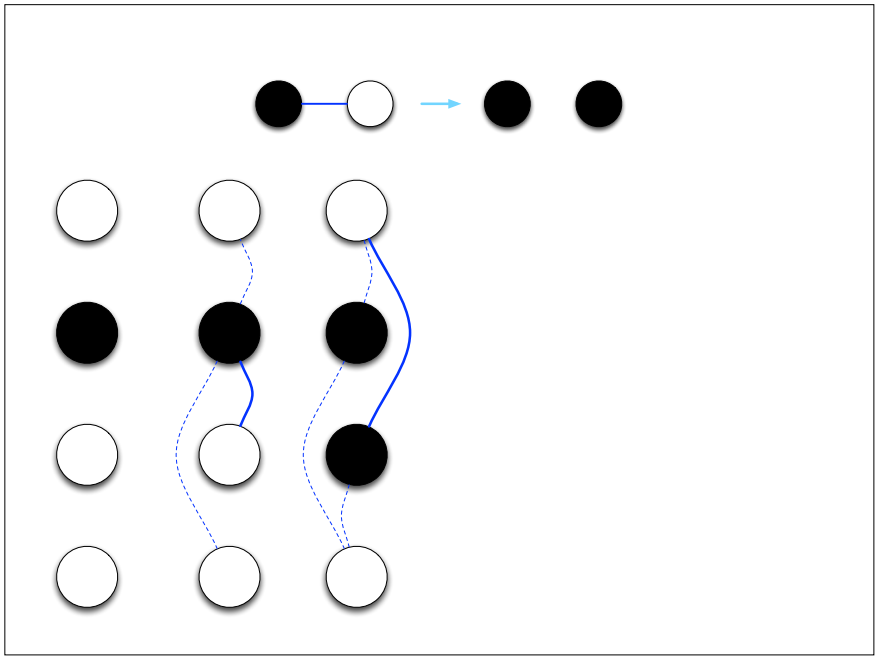
219



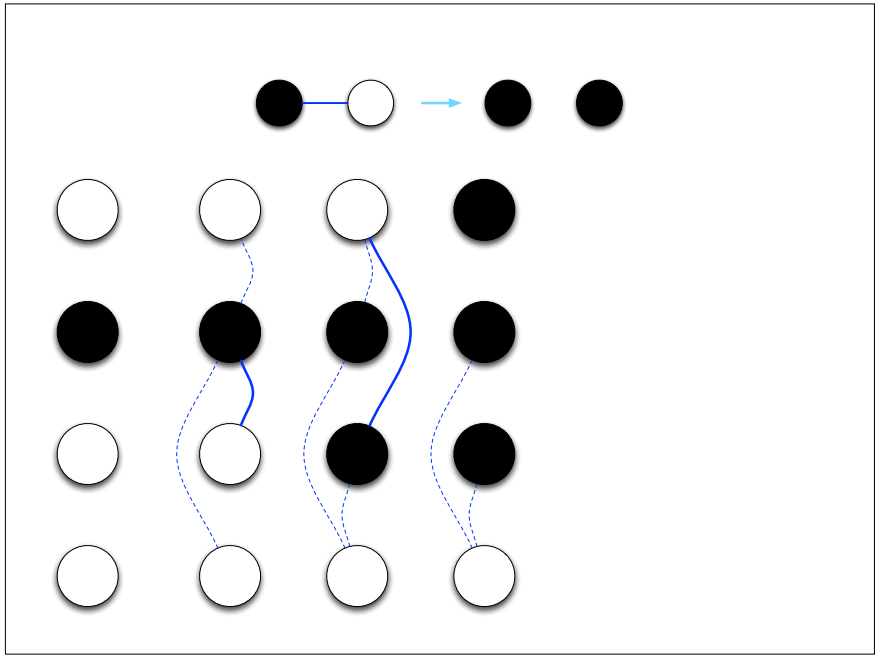
220



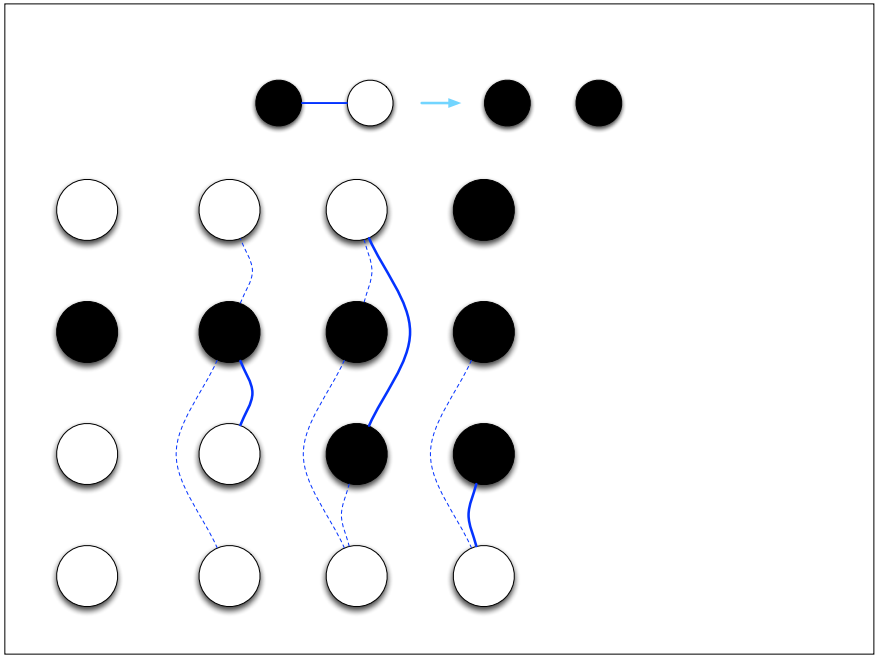
221



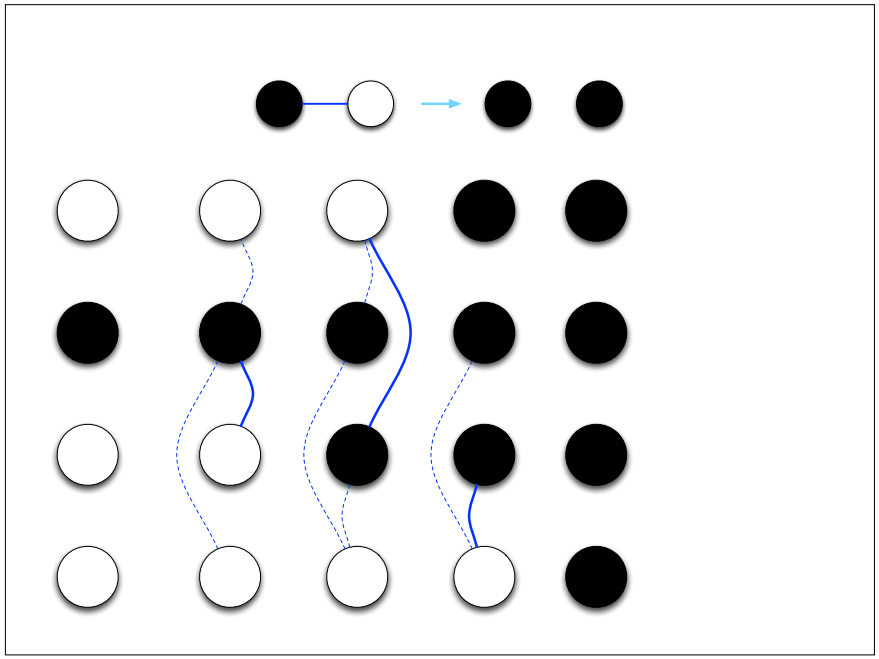
222



223



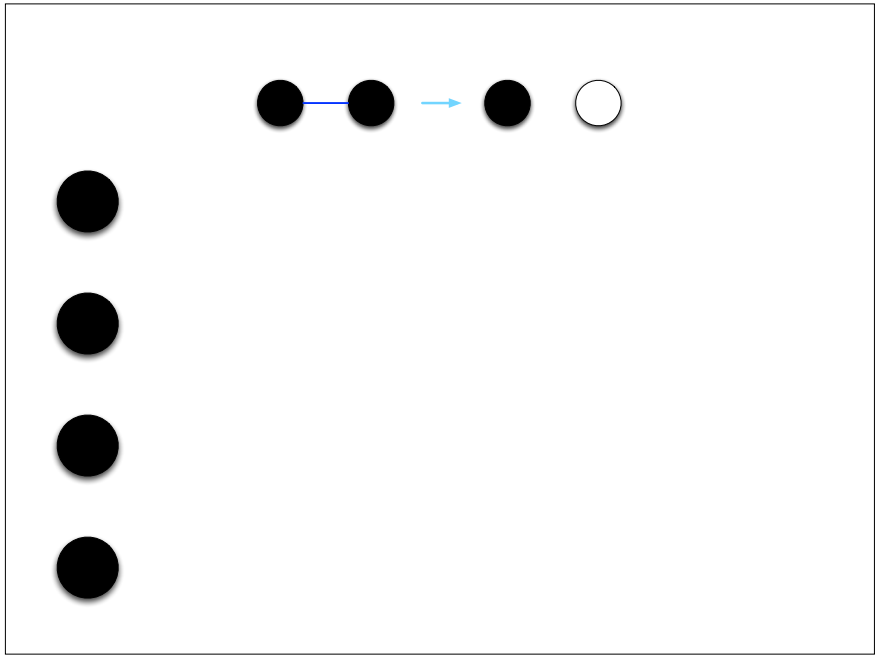
224



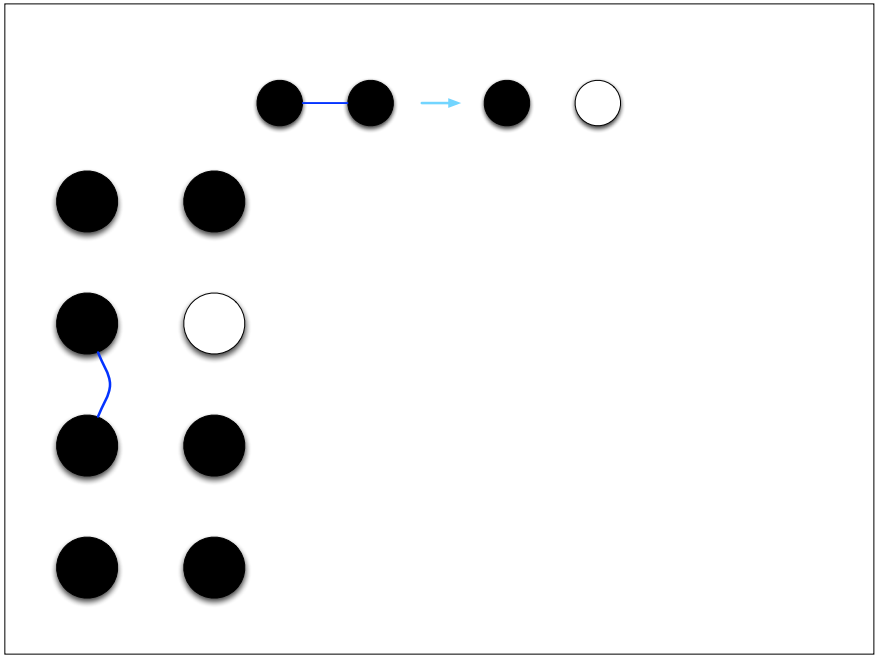
225

Example 1b

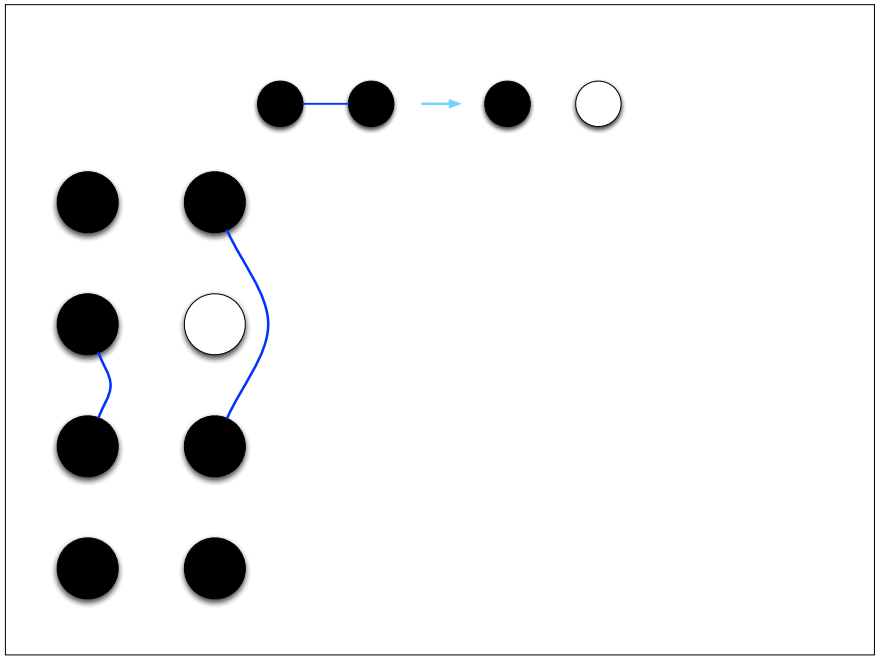
226



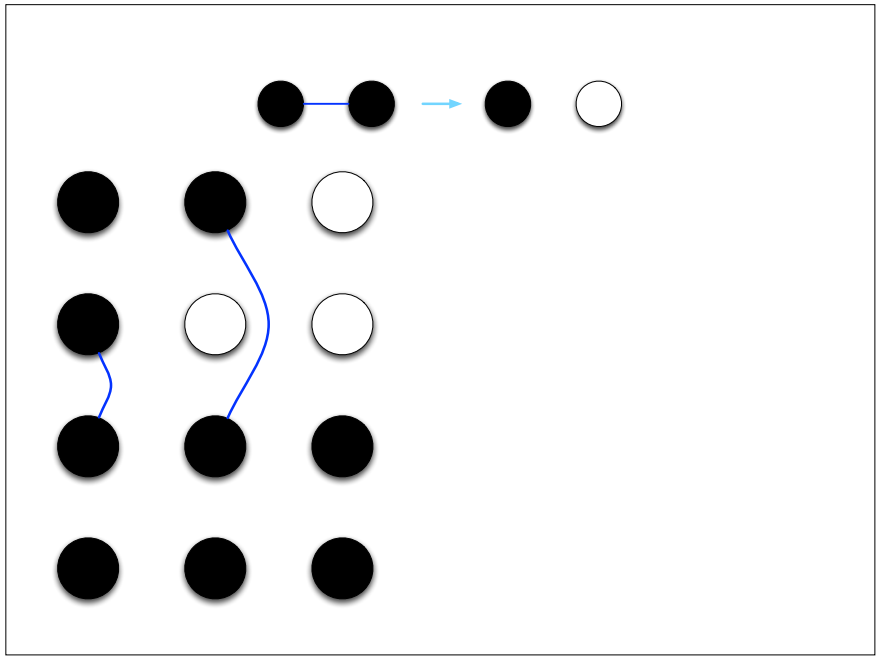
227



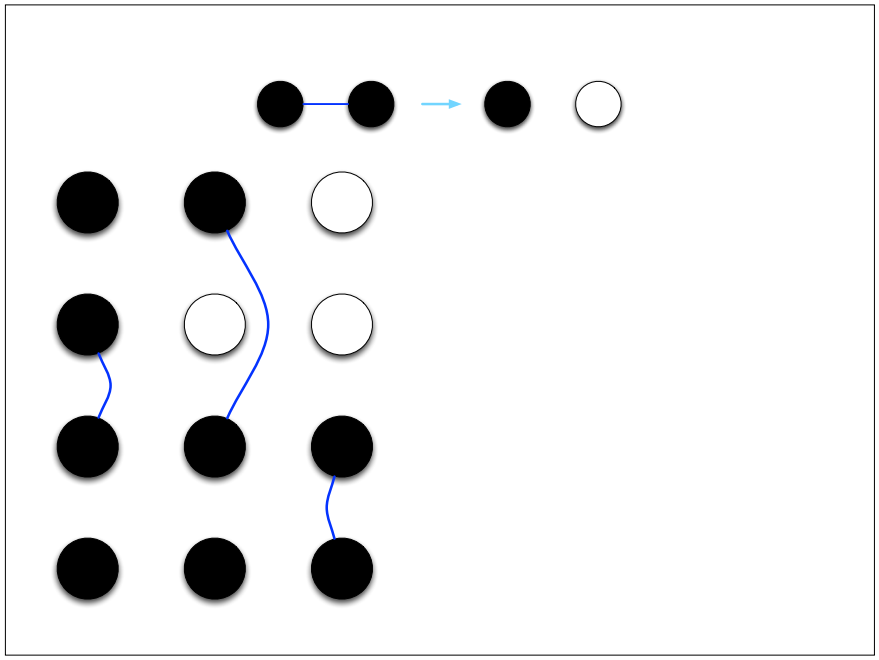
228



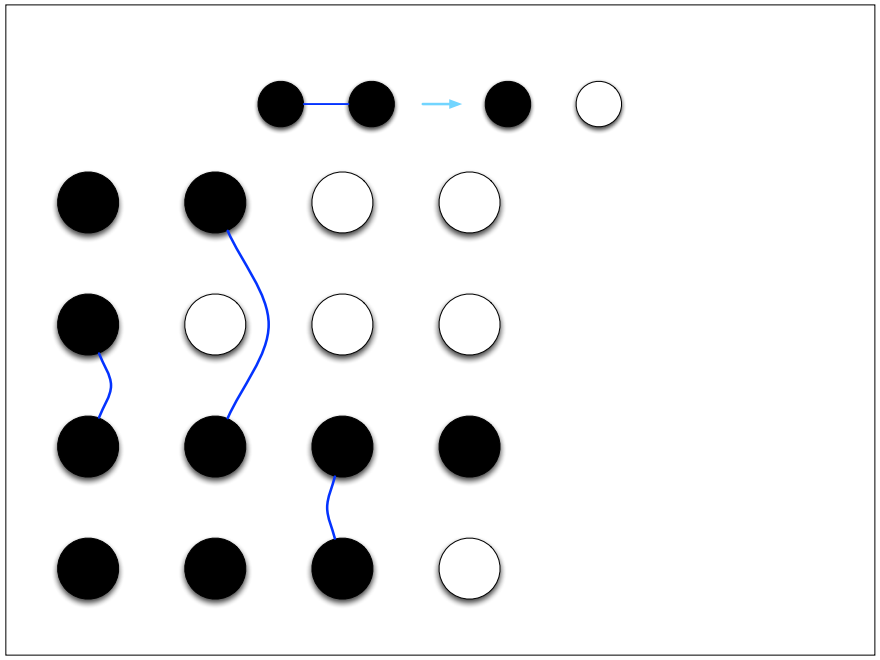
229



230



231

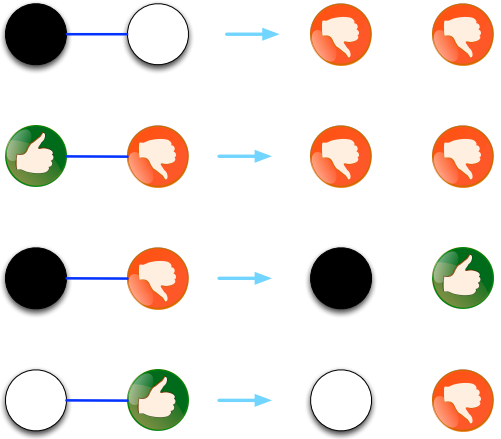


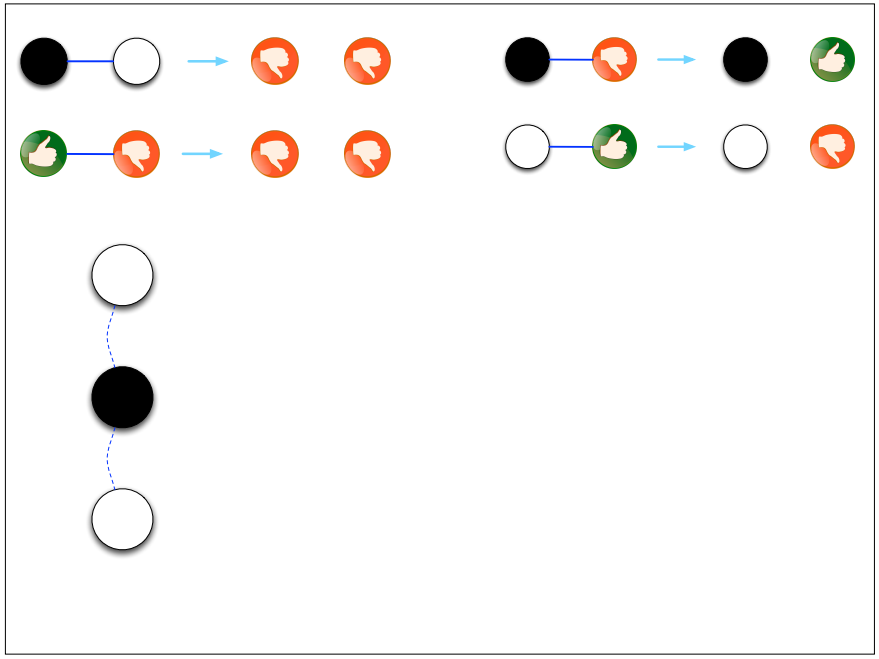
232

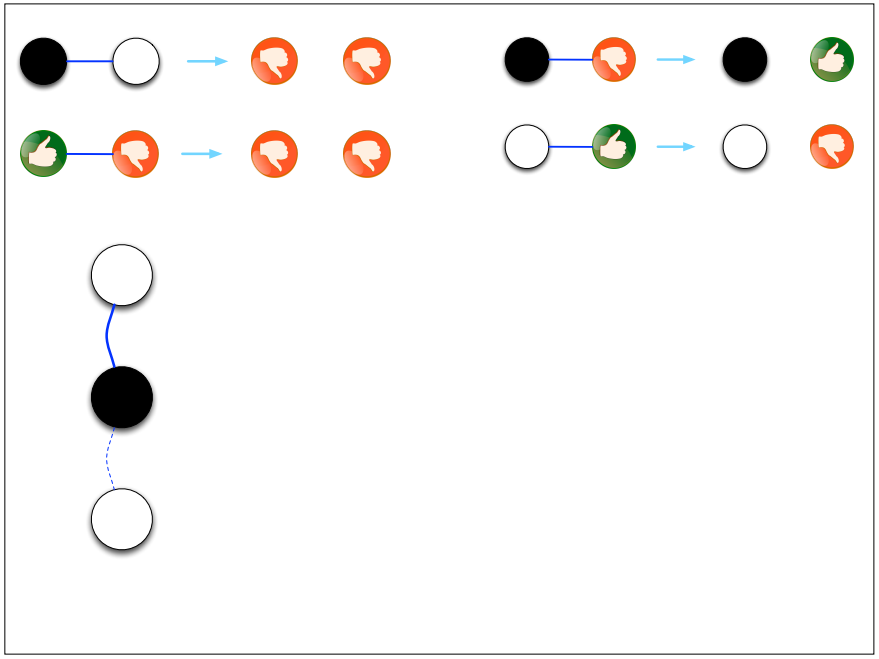
Example 2

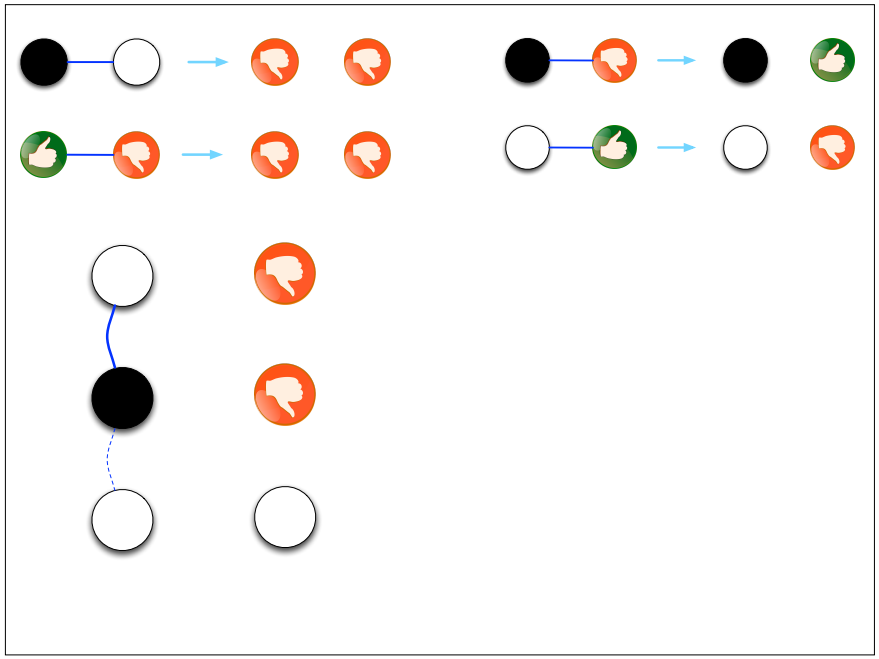
- **Inputs**:
- **Outputs**:
- **#** **> #** **?**

Example 2

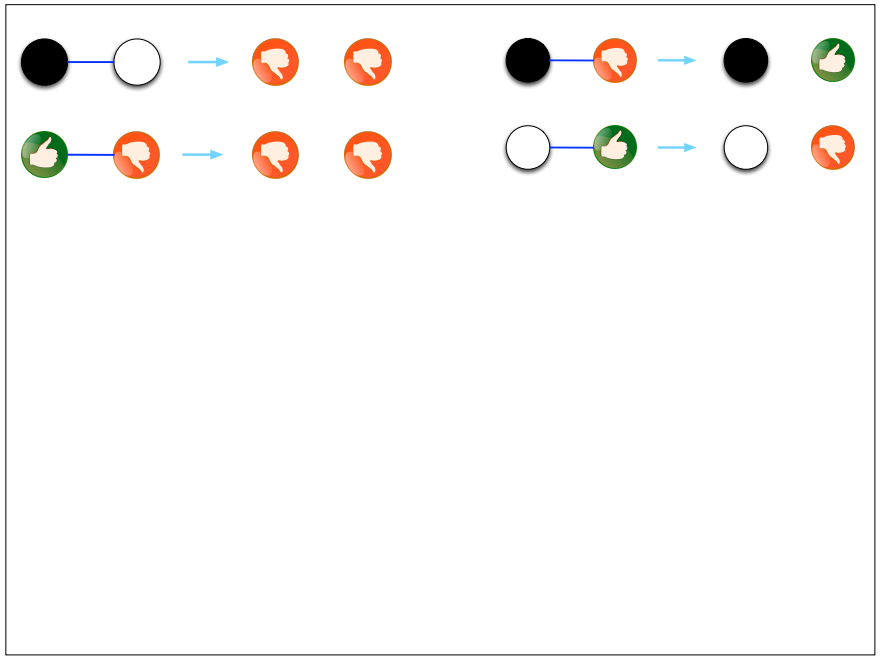




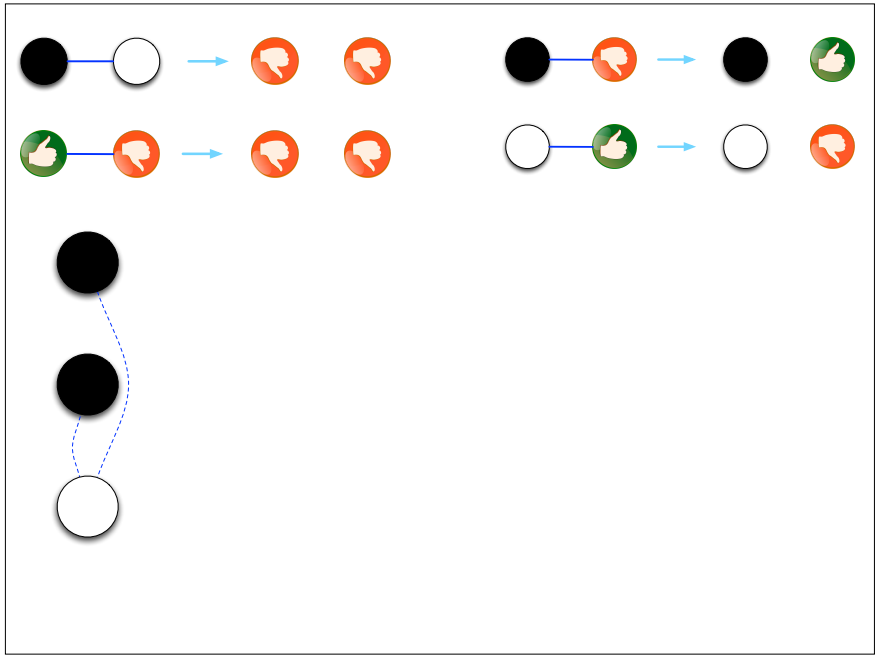




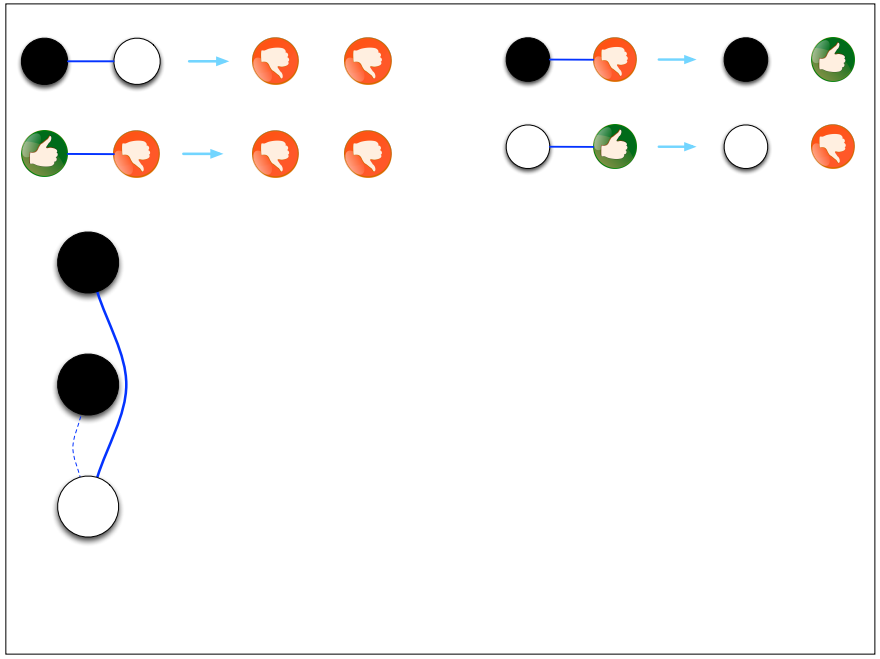
237



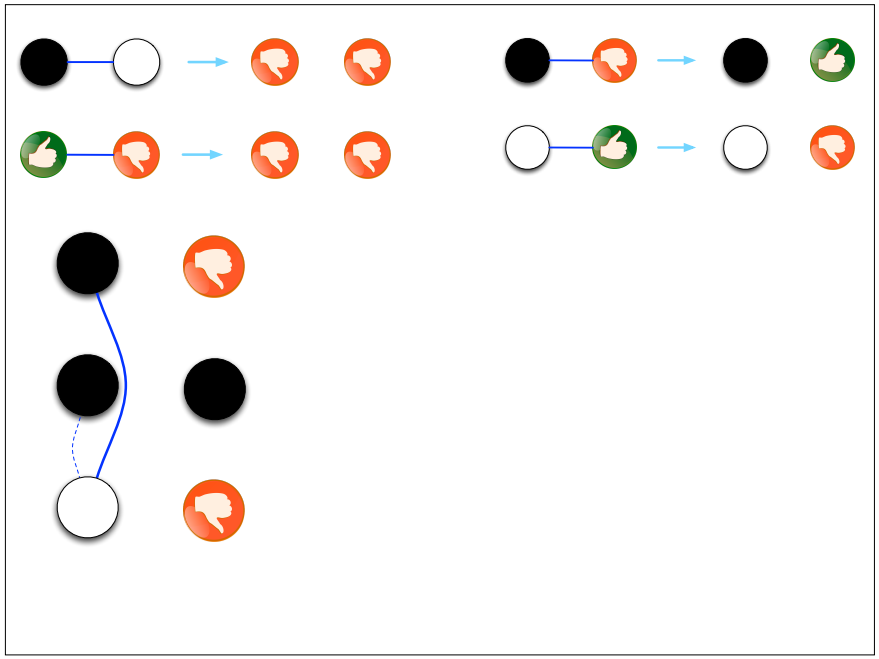
238



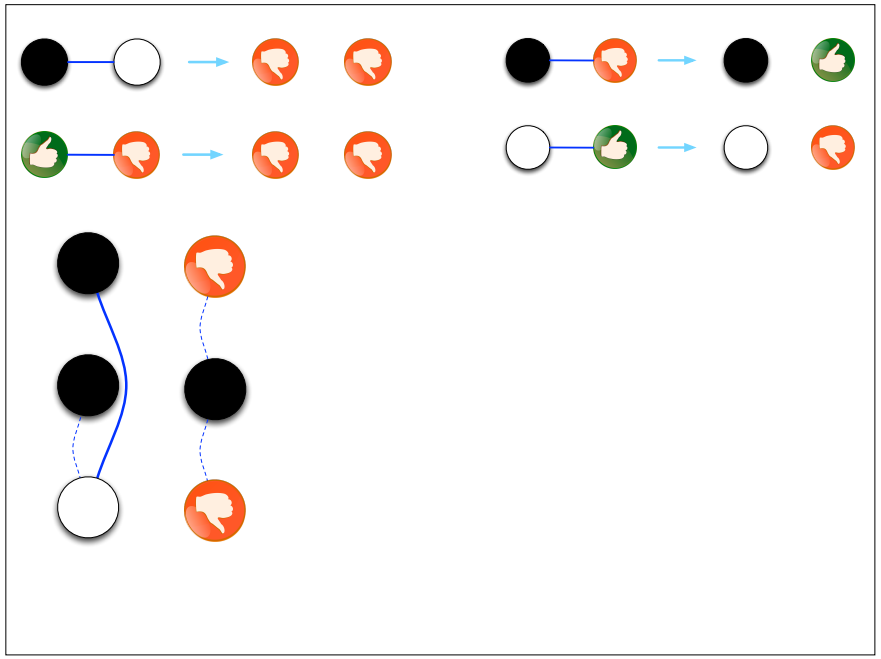
239



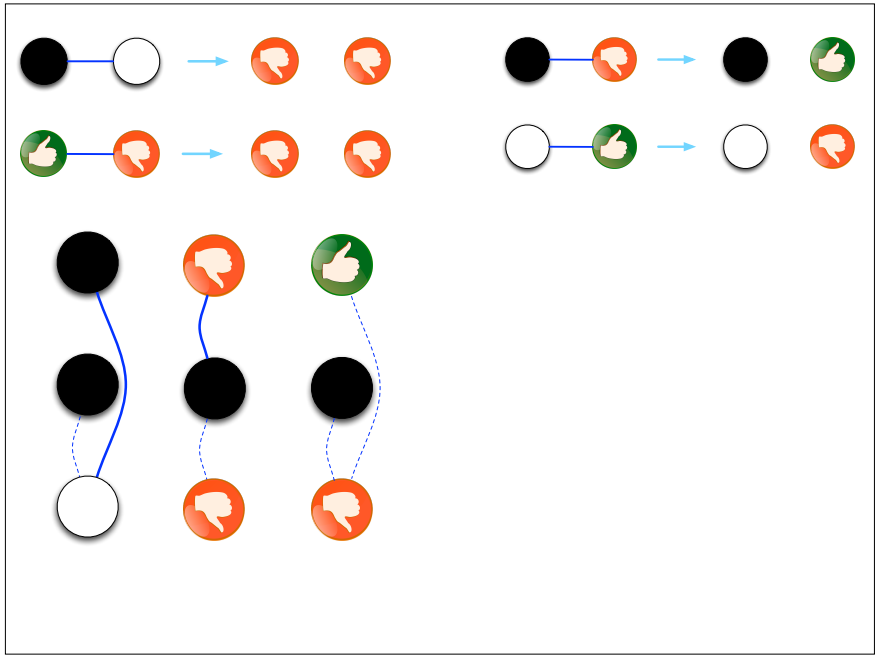
240



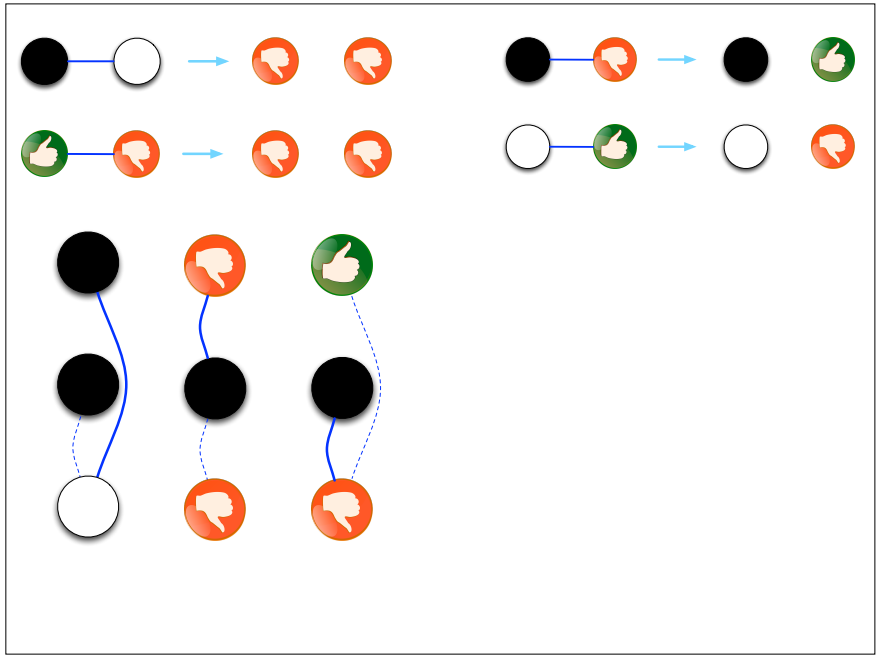
241



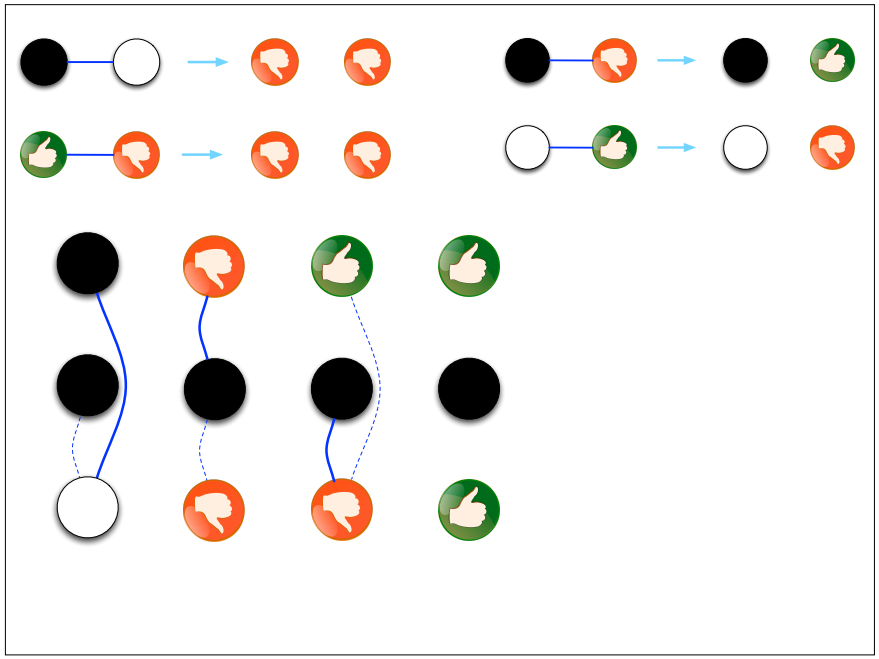
242



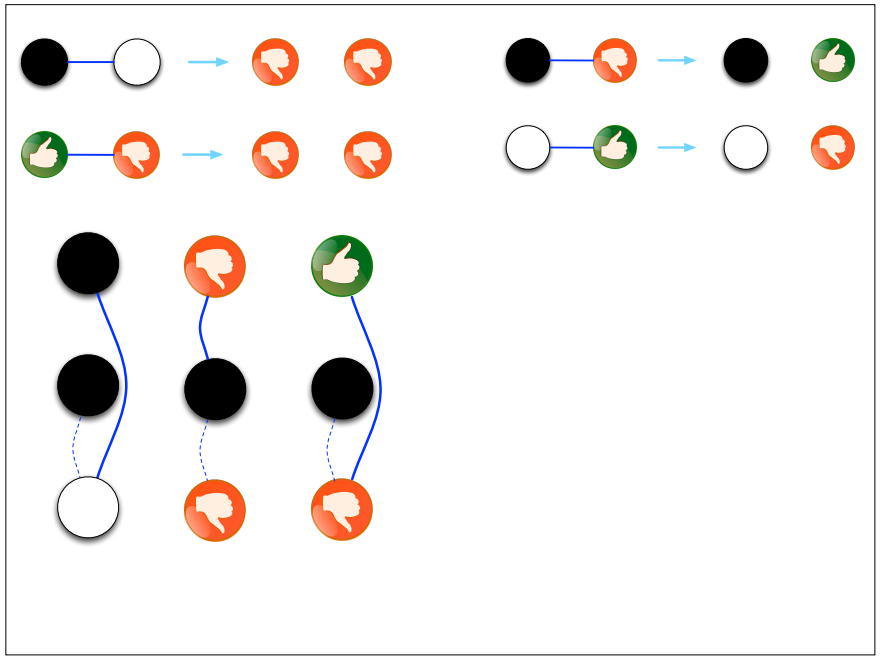
243



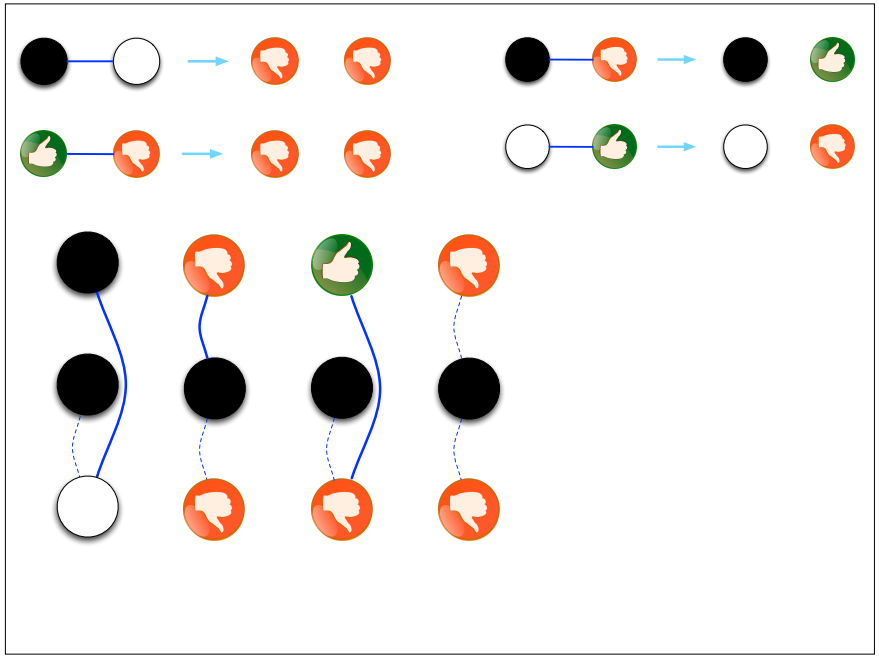
244



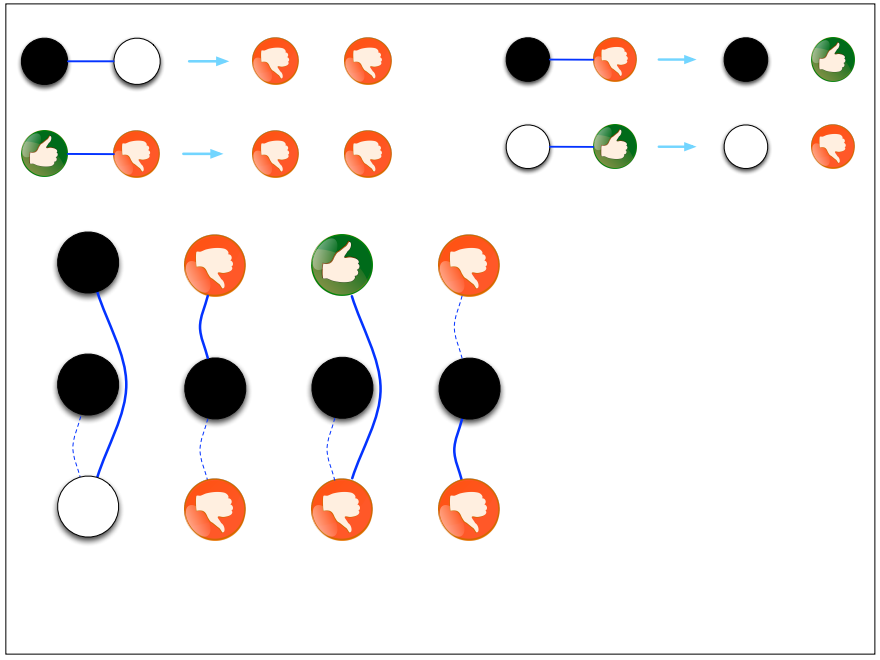
245



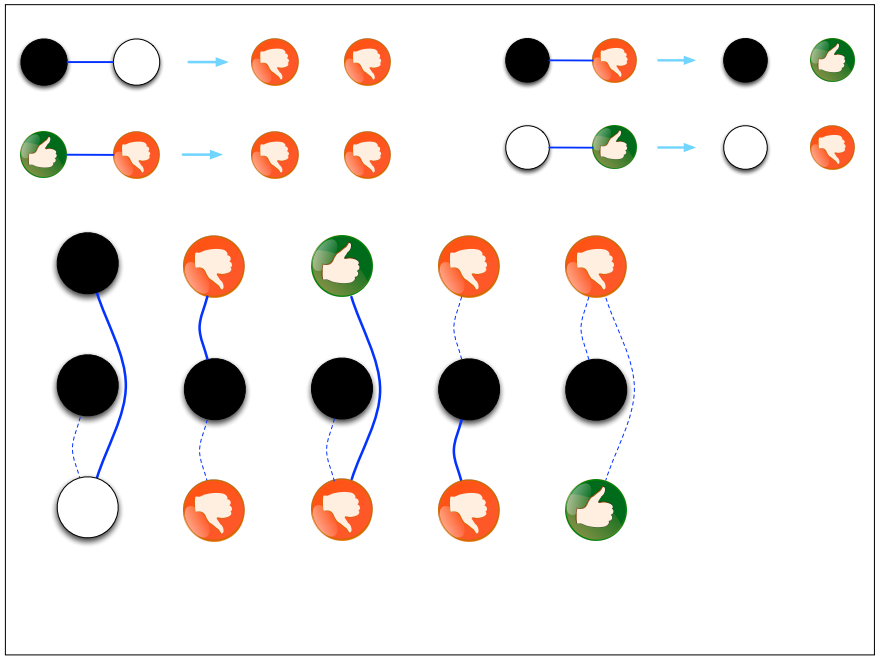
246



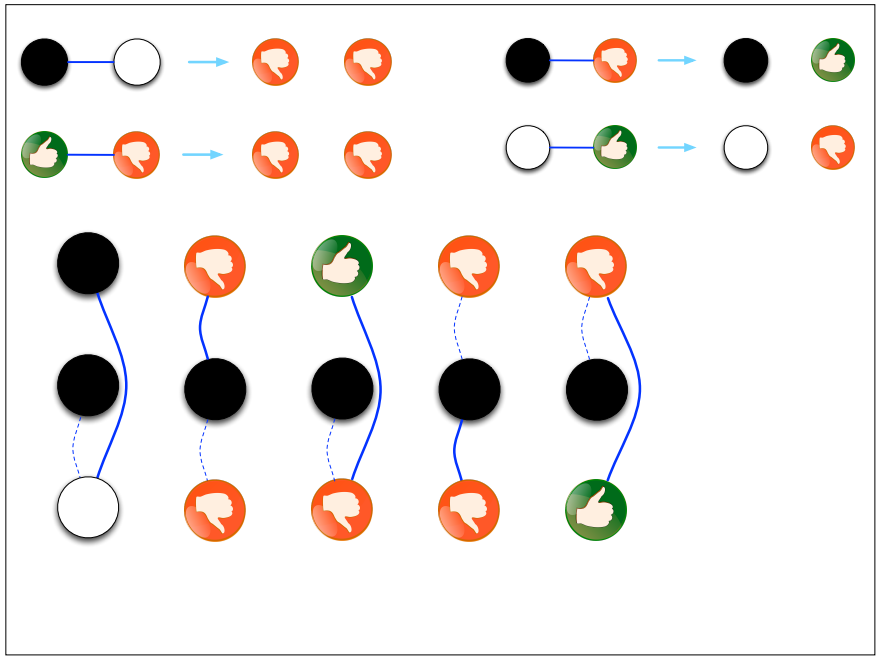
247



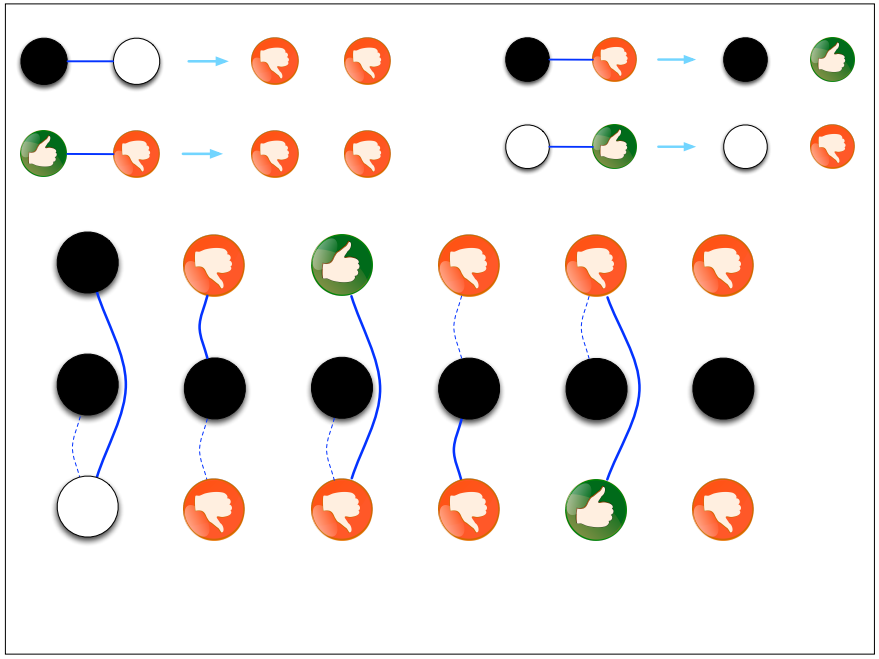
248



249



250



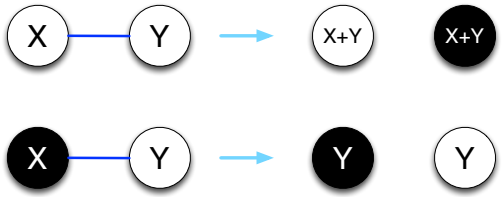
251

Example 3

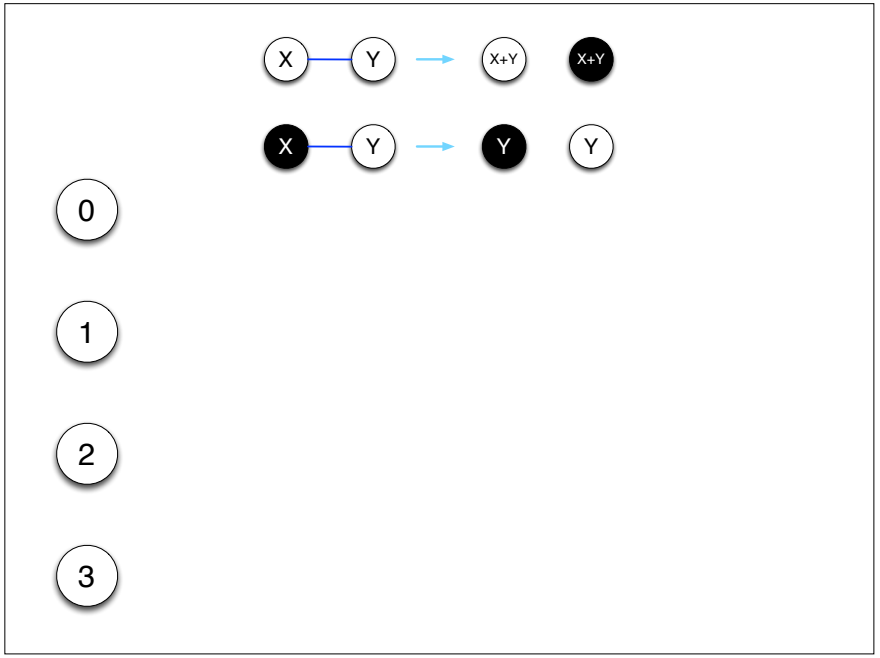
- **Inputs**: 0 1 2 3
- **Outputs**: 0 1 2 3 0 1 2 3
- **Sum mod 4 ?**

252

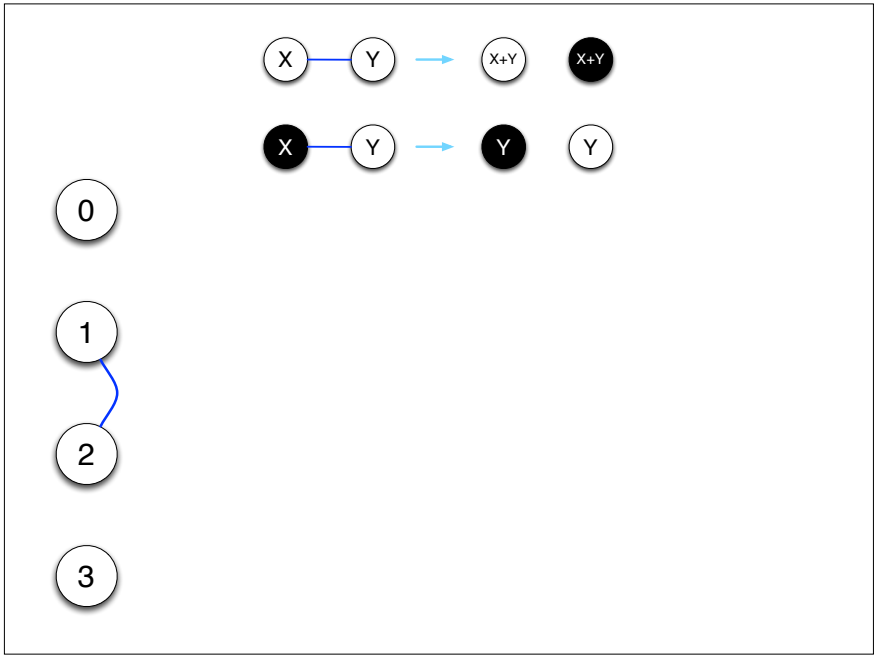
Example 3



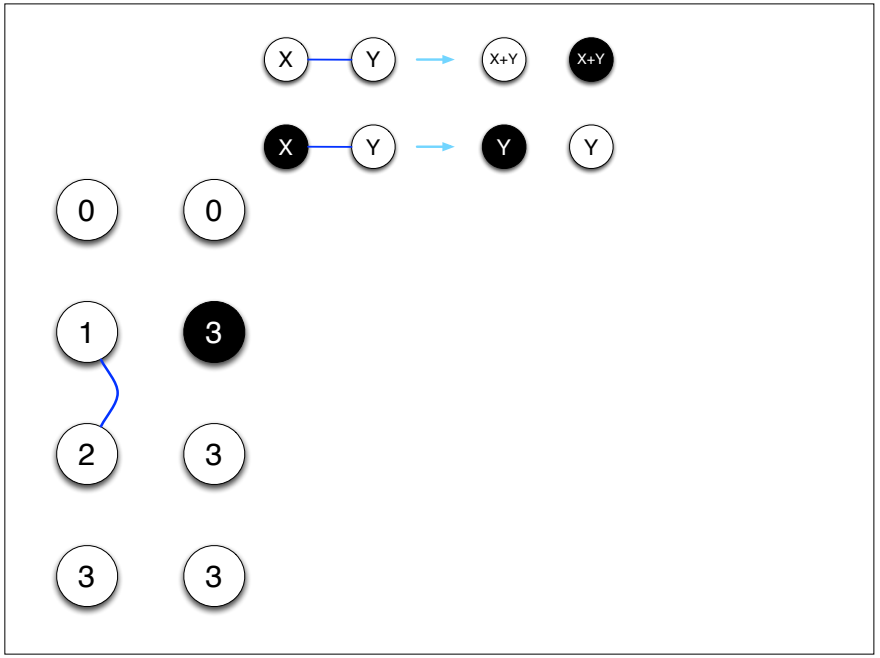
253



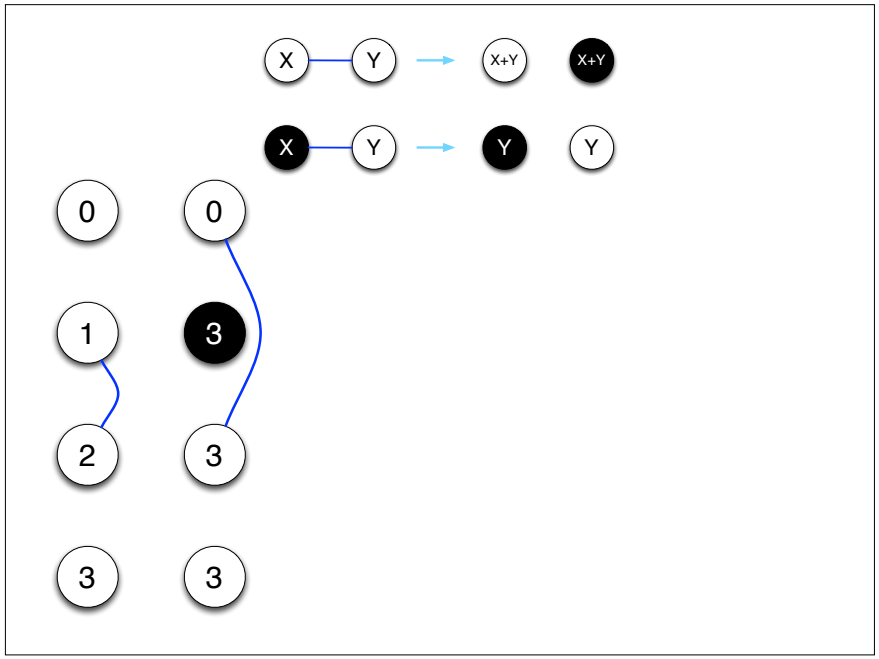
254



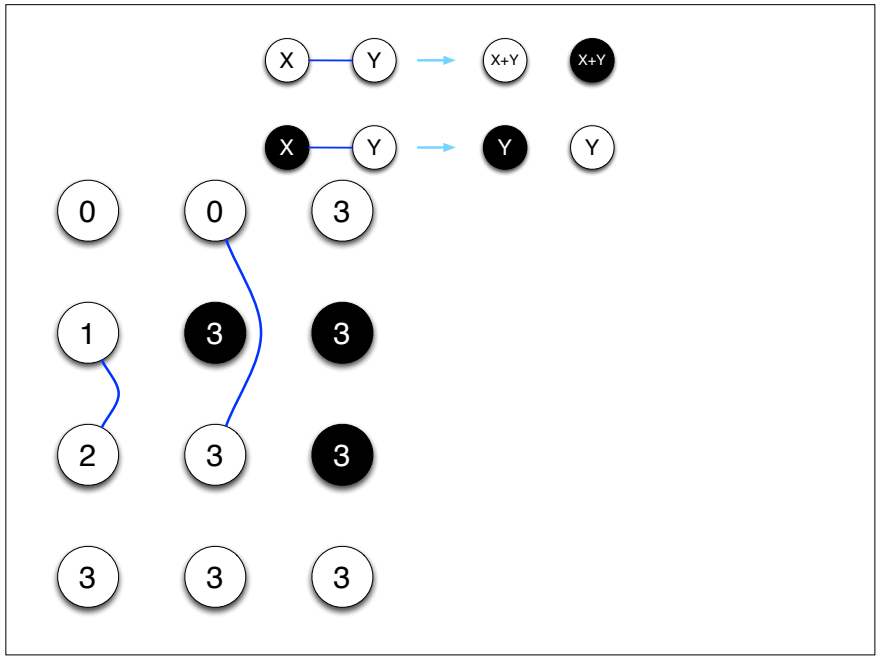
255



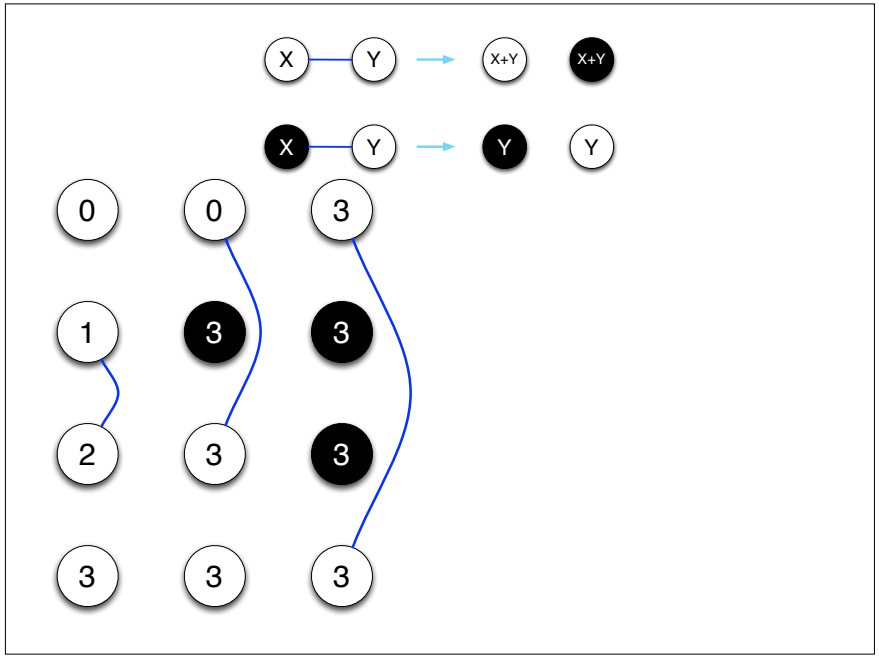
256



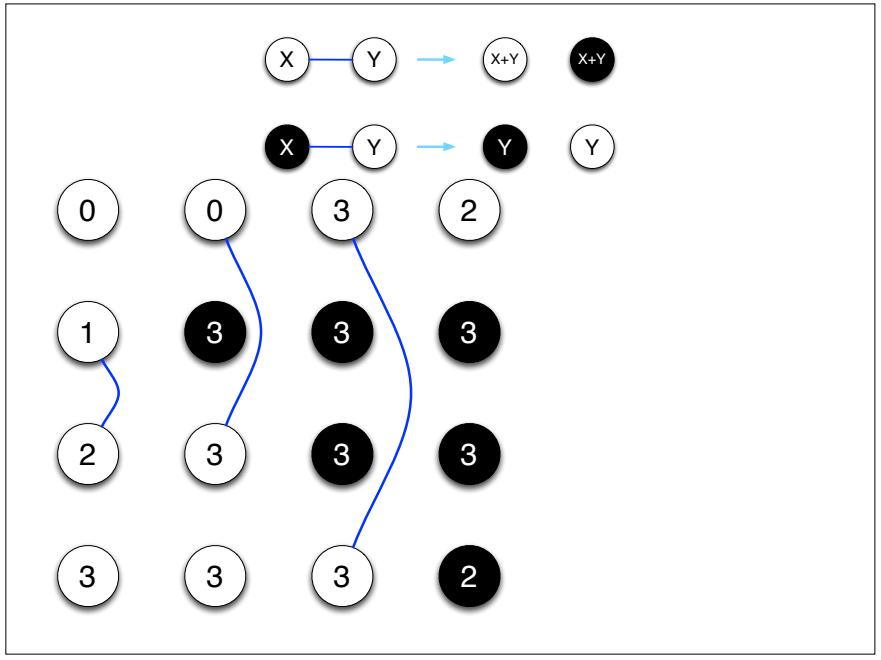
257



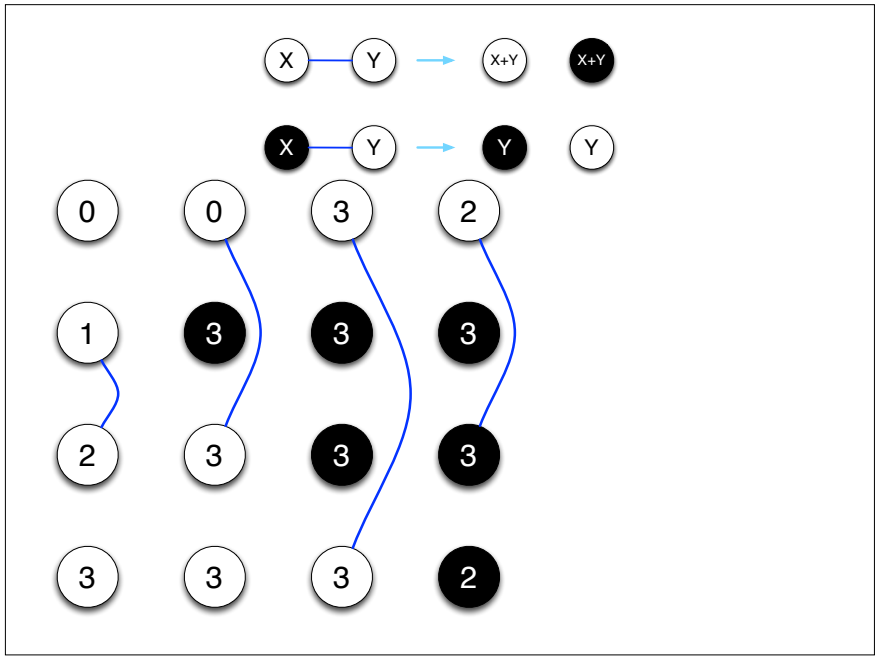
258



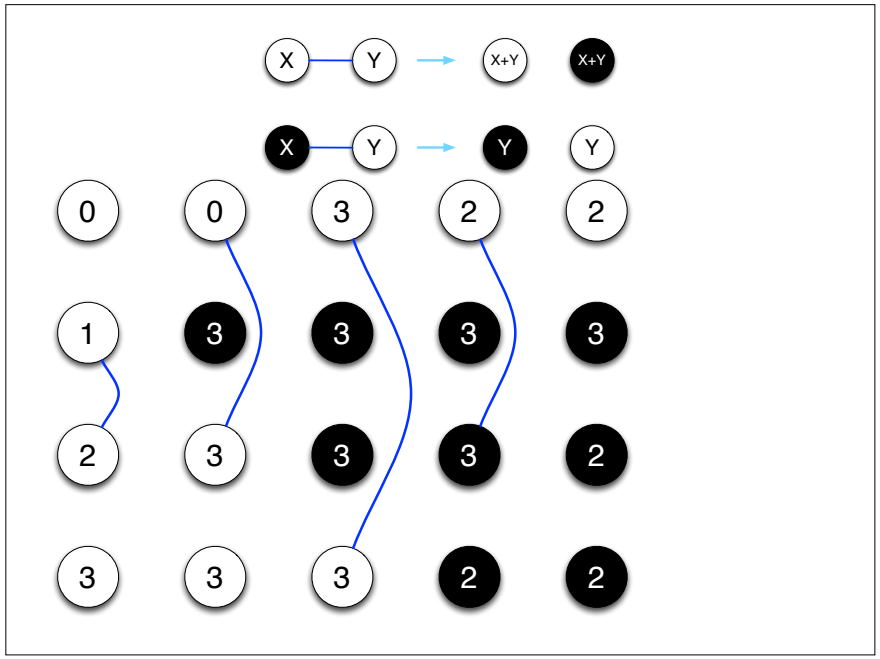
259



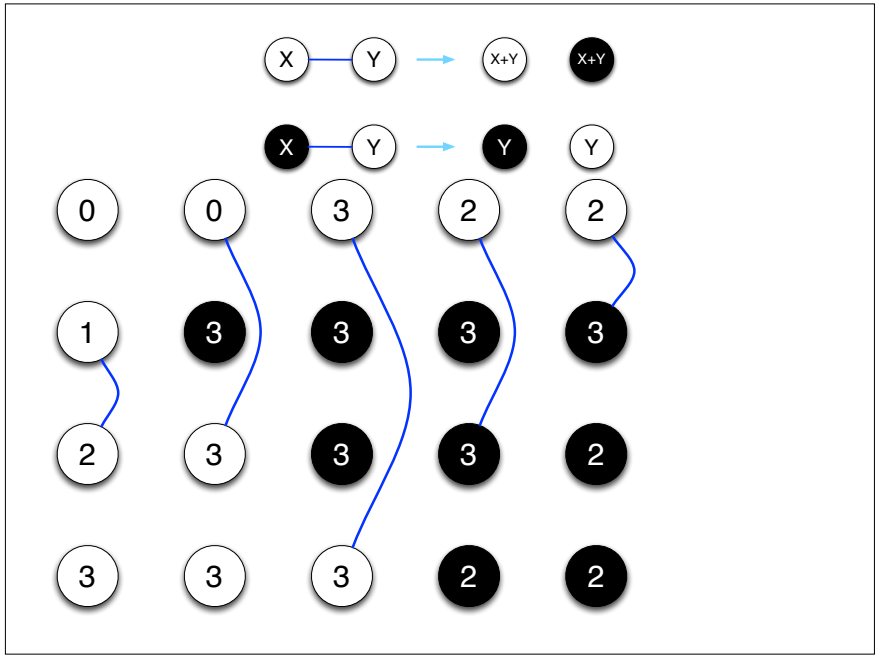
260



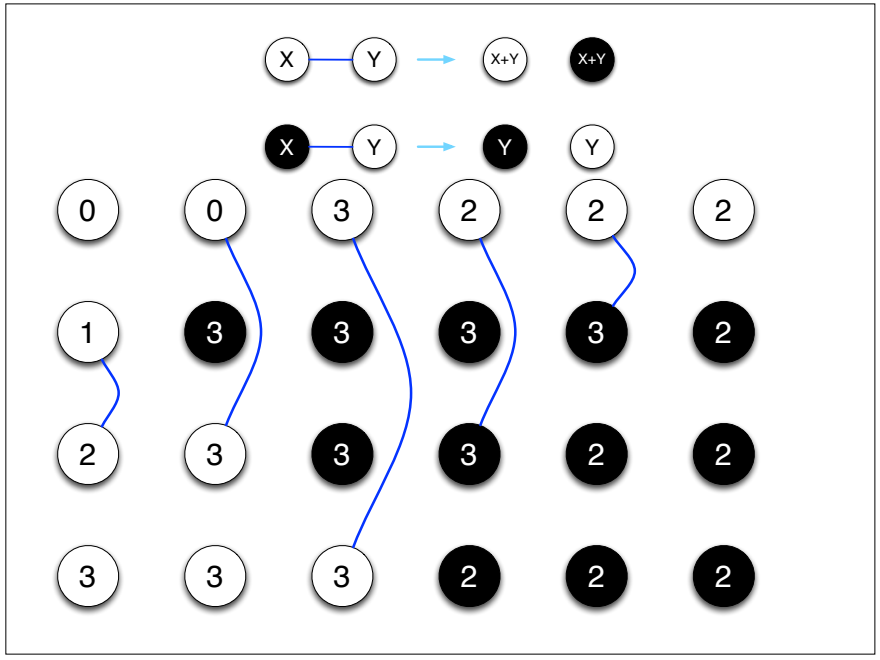
261

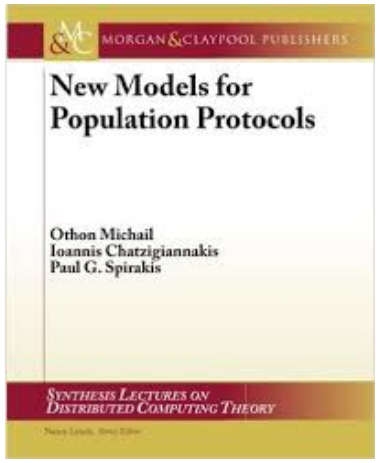


262



263



264

Population Protocols