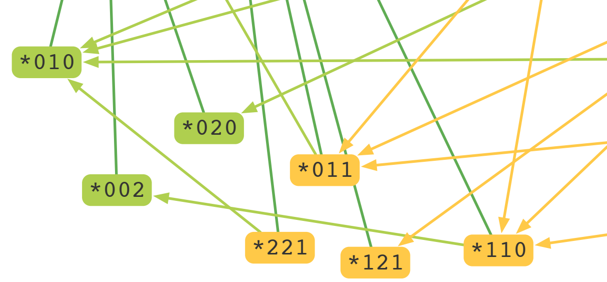
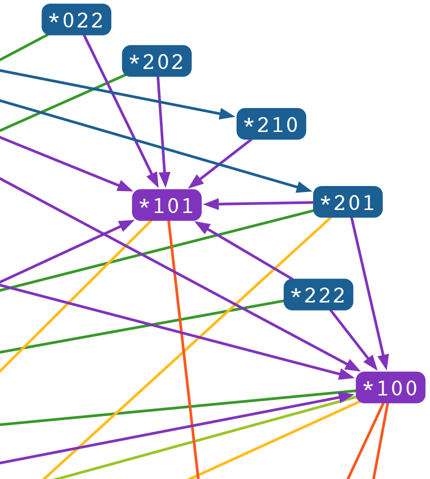


Distributed algorithms and computational algorithm design

Joel Rybicki
HIIT, University of Helsinki

September 27, 2013
HIIT seminar, Kumpula



Joint work with

Danny Dolev
The Hebrew University of Jerusalem

Christoph Lenzen
MIT

Juho Hirvonen
Janne H. Korhonen
Jukka Suomela
HIIT and University of Helsinki

Computational algorithm design

Algorithm design

Ask the computer scientist:

*"Is there an algorithm **A** for problem **P**?"*

Algorithm design

Ask the computer ~~scientist~~:

"Is there an algorithm ***A*** *for problem* ***P****?"*

Computational algorithm design

Ask the computer:
"Is there an algorithm ***A*** *for problem* ***P****?"*

Searching for an algorithm

- The search space is infinite
- What if there are no algorithms?

Finite search

How to make the search space finite?

- Add resource bounds: time, memory, etc.
- Restrict the class of inputs
- Restrict the model of computation

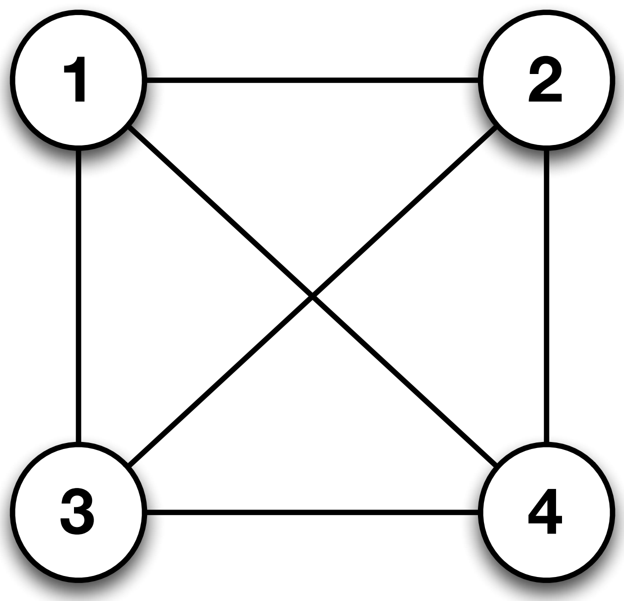
An inductive approach

Computers are good at boring calculations.
People are good at generalizing.

- Solve a (difficult) base case
- Use this to solve a more general problem

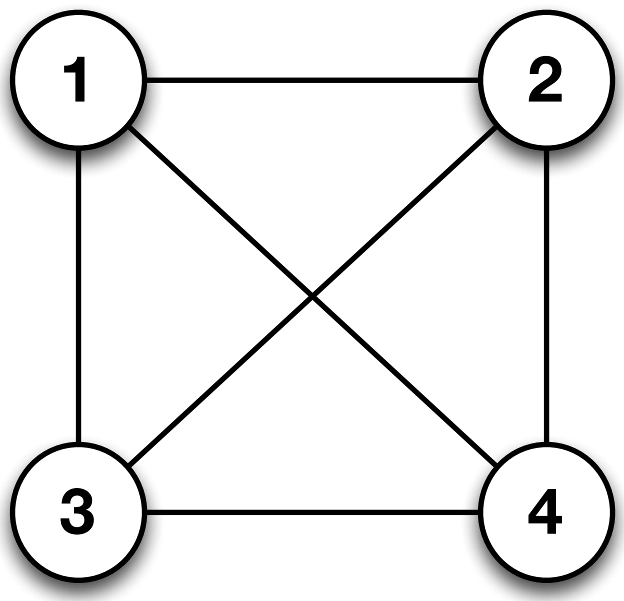
Synchronous counting

The model



- n processors
- s states
- arbitrary initial state

The model



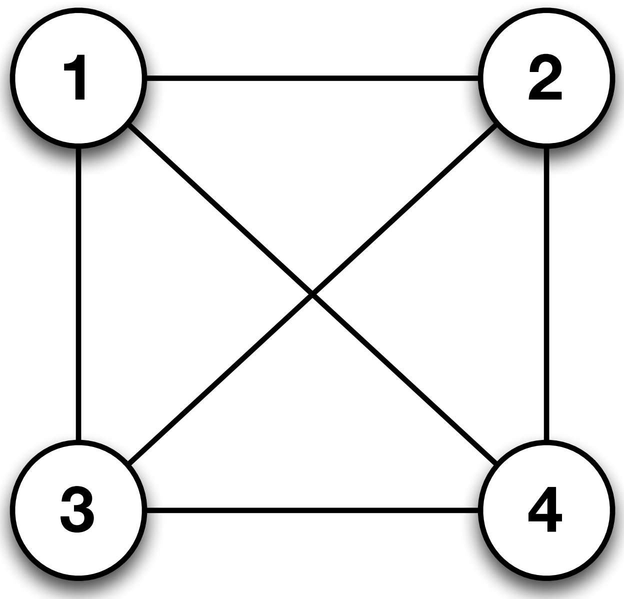
- n processors
- s states
- arbitrary initial state

Synchronous step:

1. broadcast state

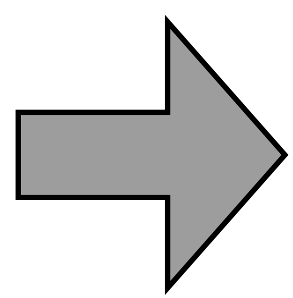
2. update state

The model



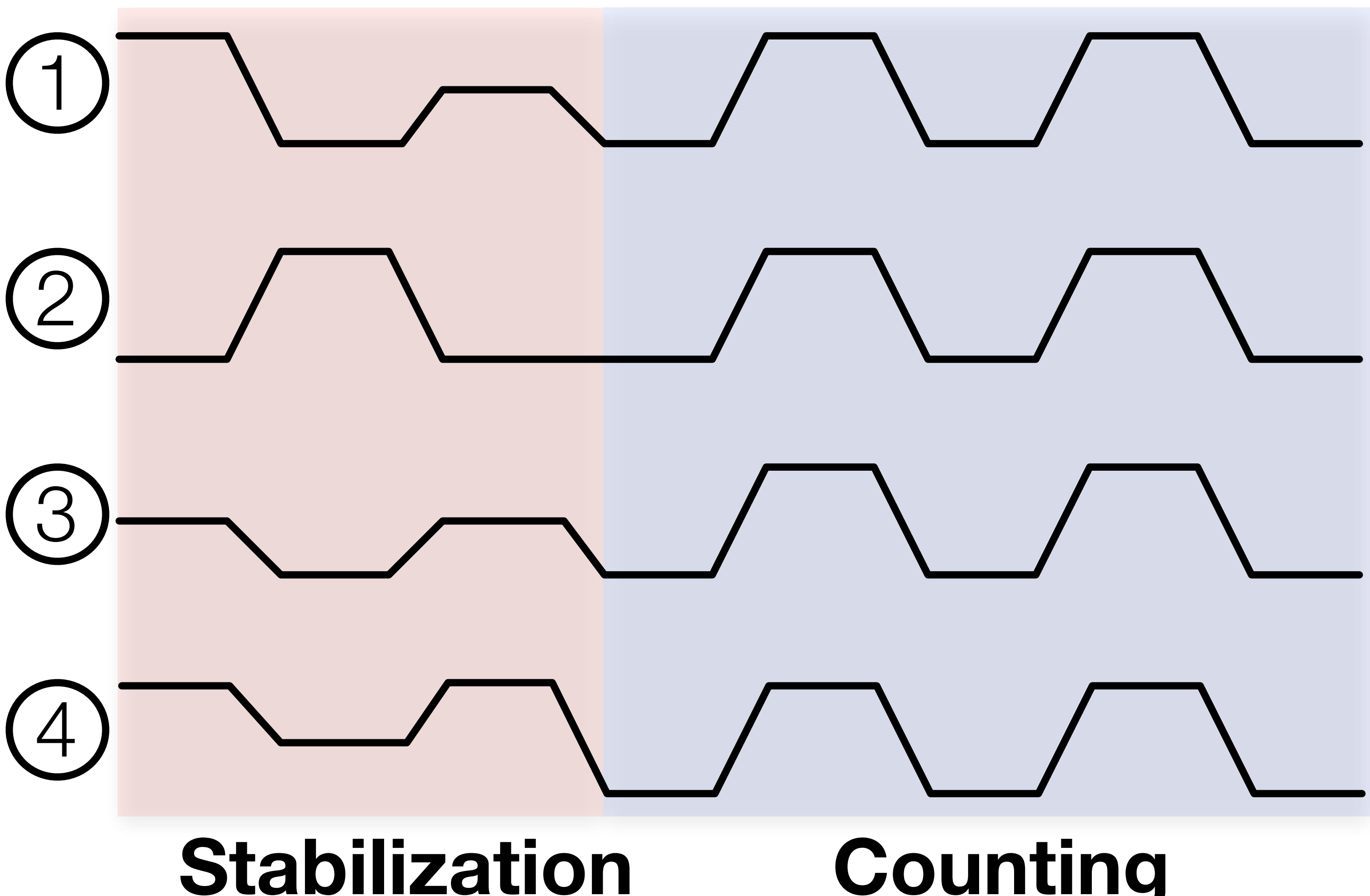
- *n* processors
- *s* states
- arbitrary initial state

Synchronous step:
1. broadcast state
2. update state



algorithm
=
transition function

Self-stabilizing counting



Self-stabilizing counting

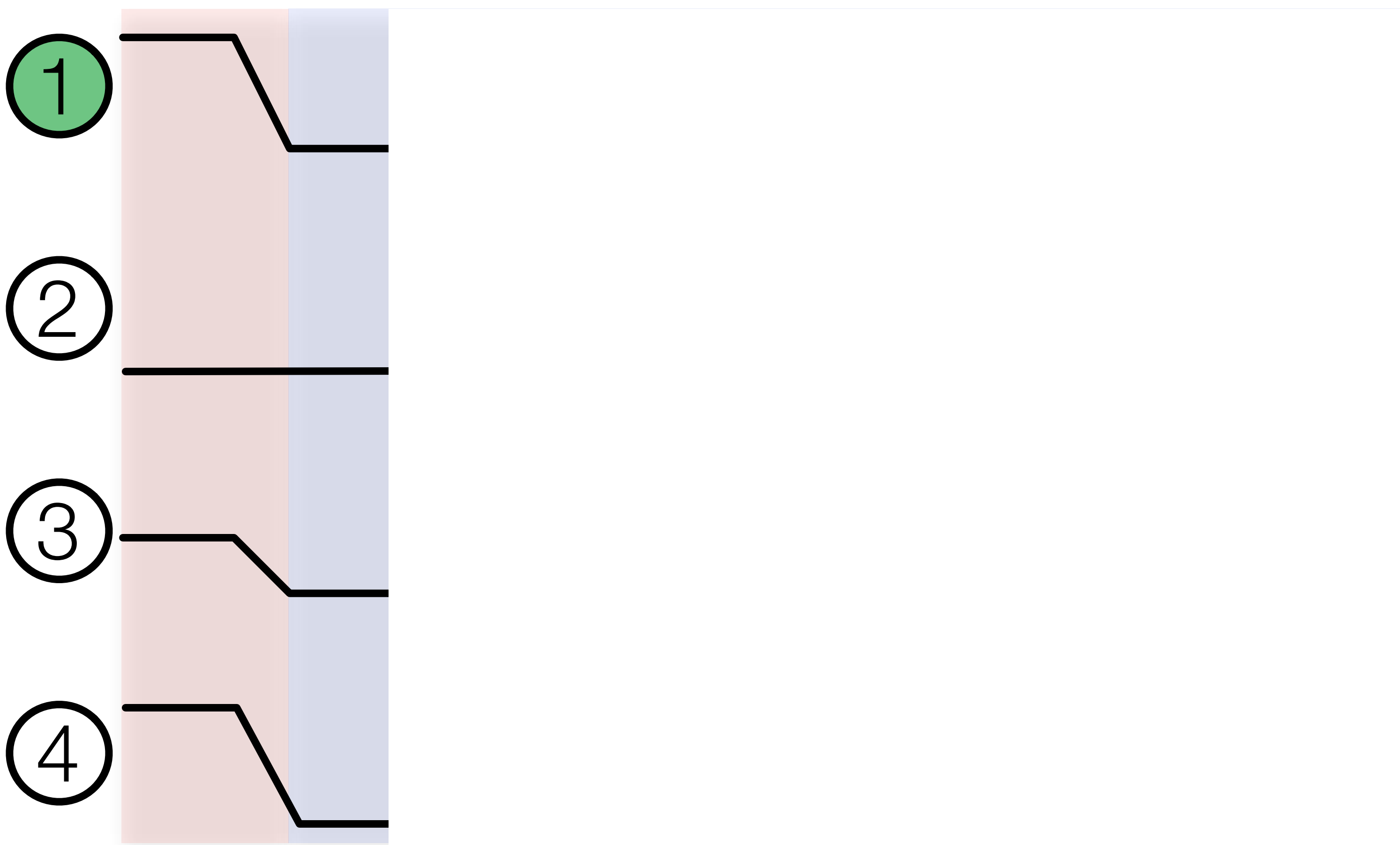
A simple algorithm solves the problem

Self-stabilizing counting

Solution: Follow the leader.

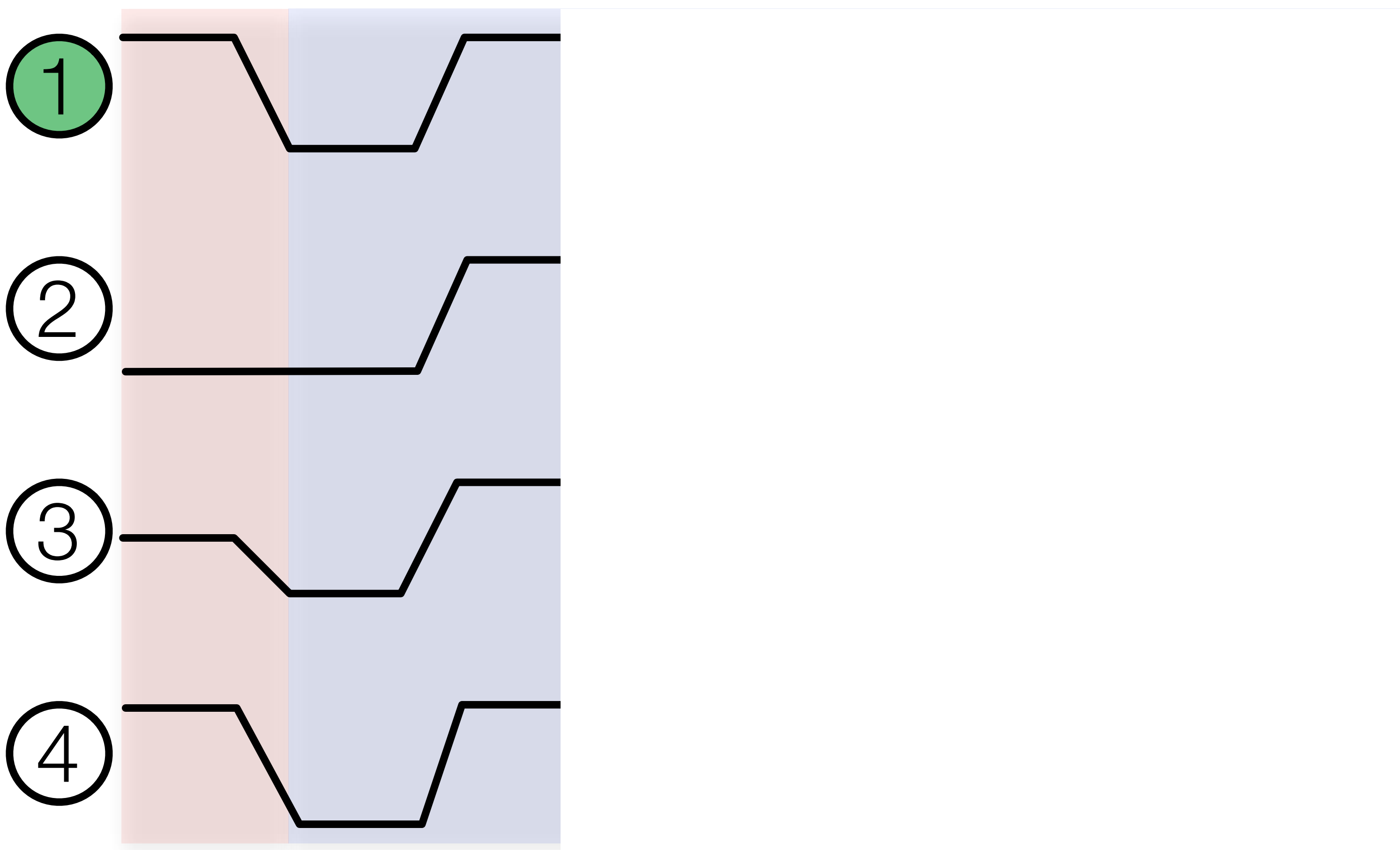
Self-stabilizing counting

Solution: Follow the leader.



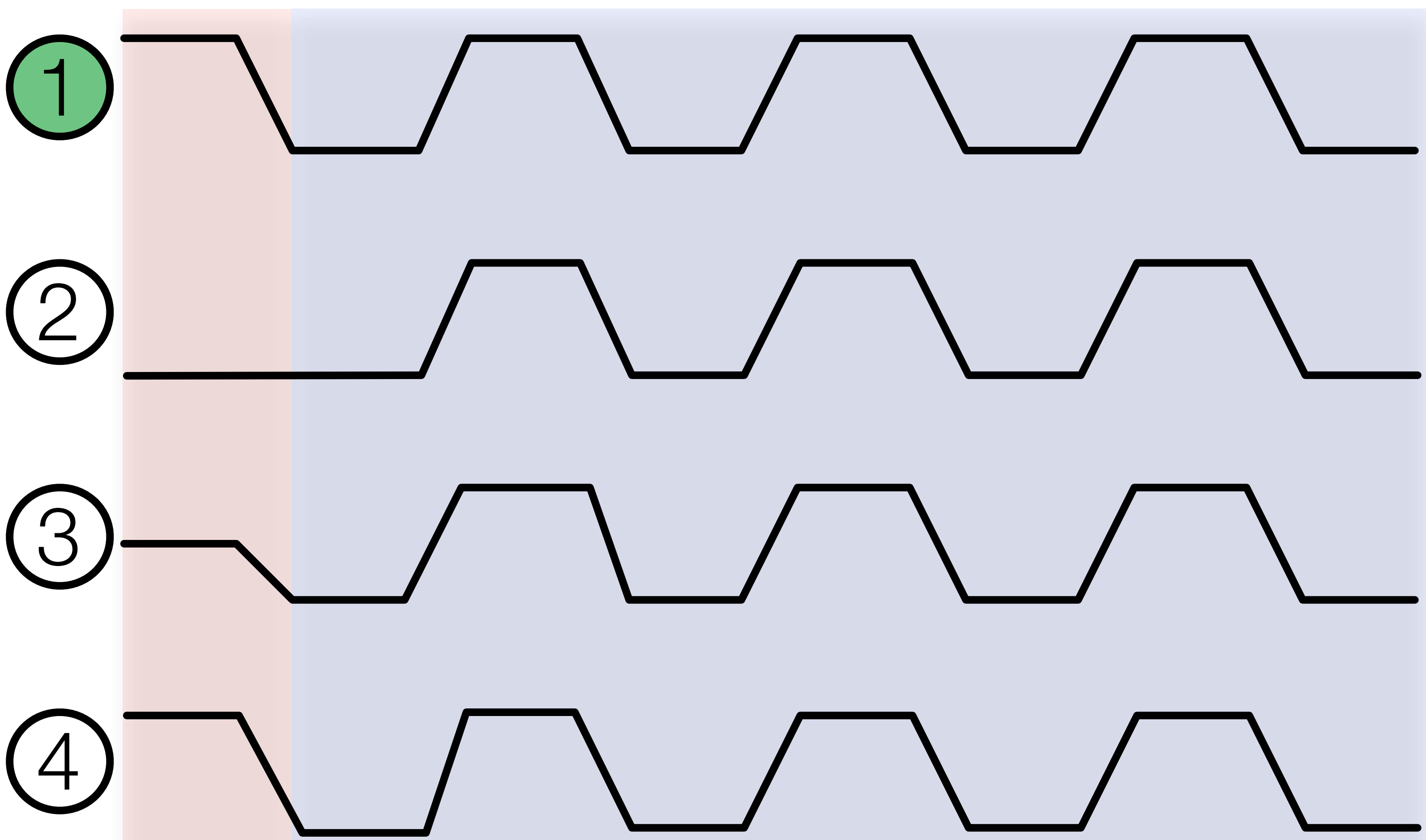
Self-stabilizing counting

Solution: Follow the leader.

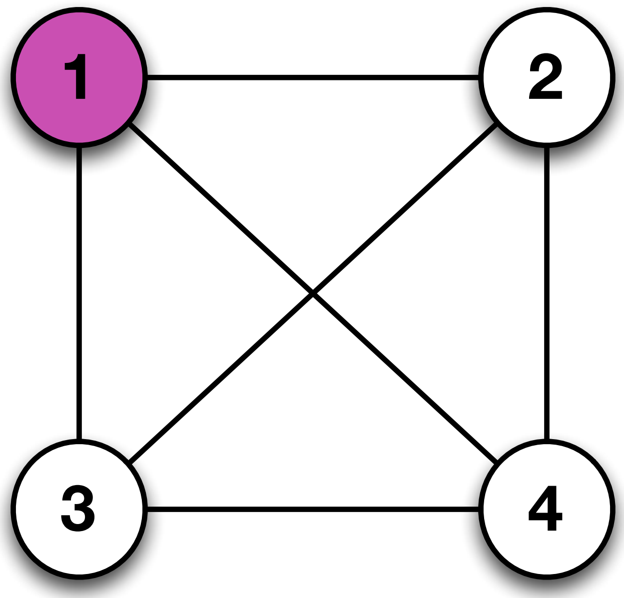


Self-stabilizing counting

Solution: Follow the leader.

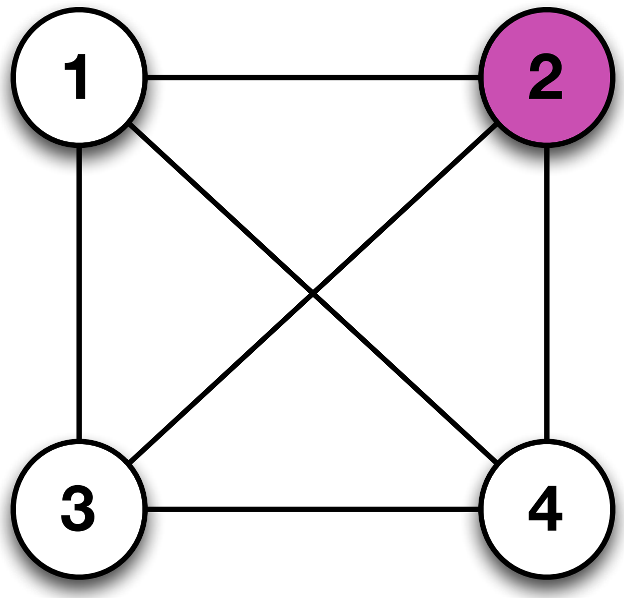


Tolerating Byzantine failures



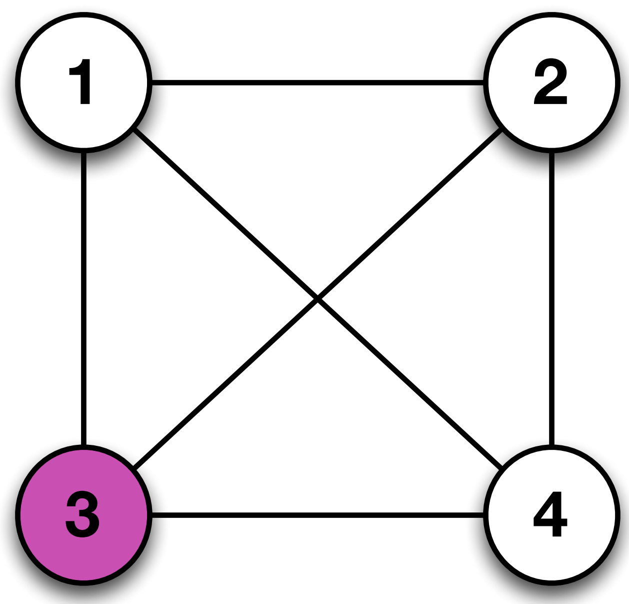
Assume that some nodes may be *Byzantine*.

Tolerating Byzantine failures



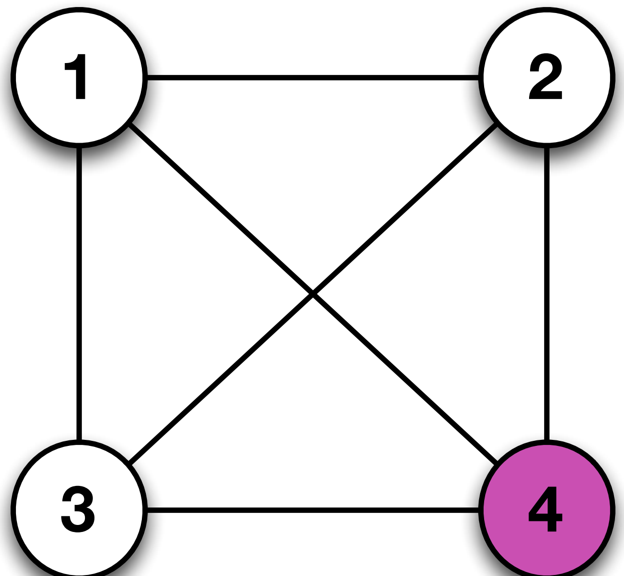
Assume that some nodes may be *Byzantine*.

Tolerating Byzantine failures



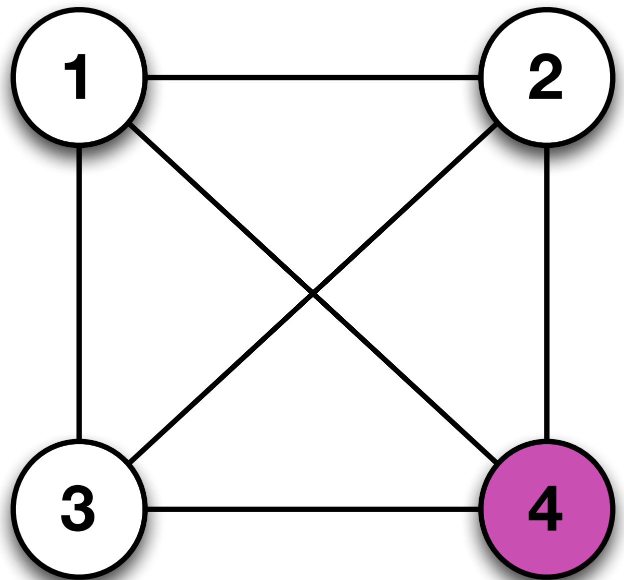
Assume that some nodes may be *Byzantine*.

Tolerating Byzantine failures



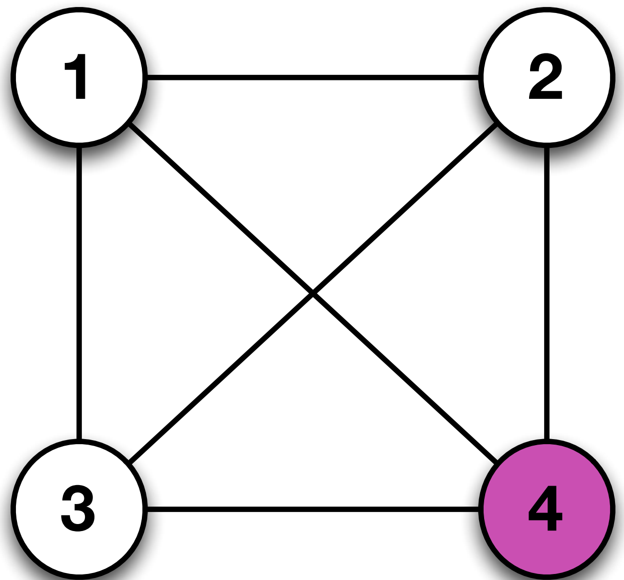
Assume that some nodes may be *Byzantine*.

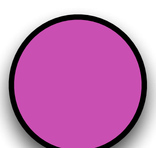
Tolerating Byzantine failures



can send *different* messages to non-faulty nodes!

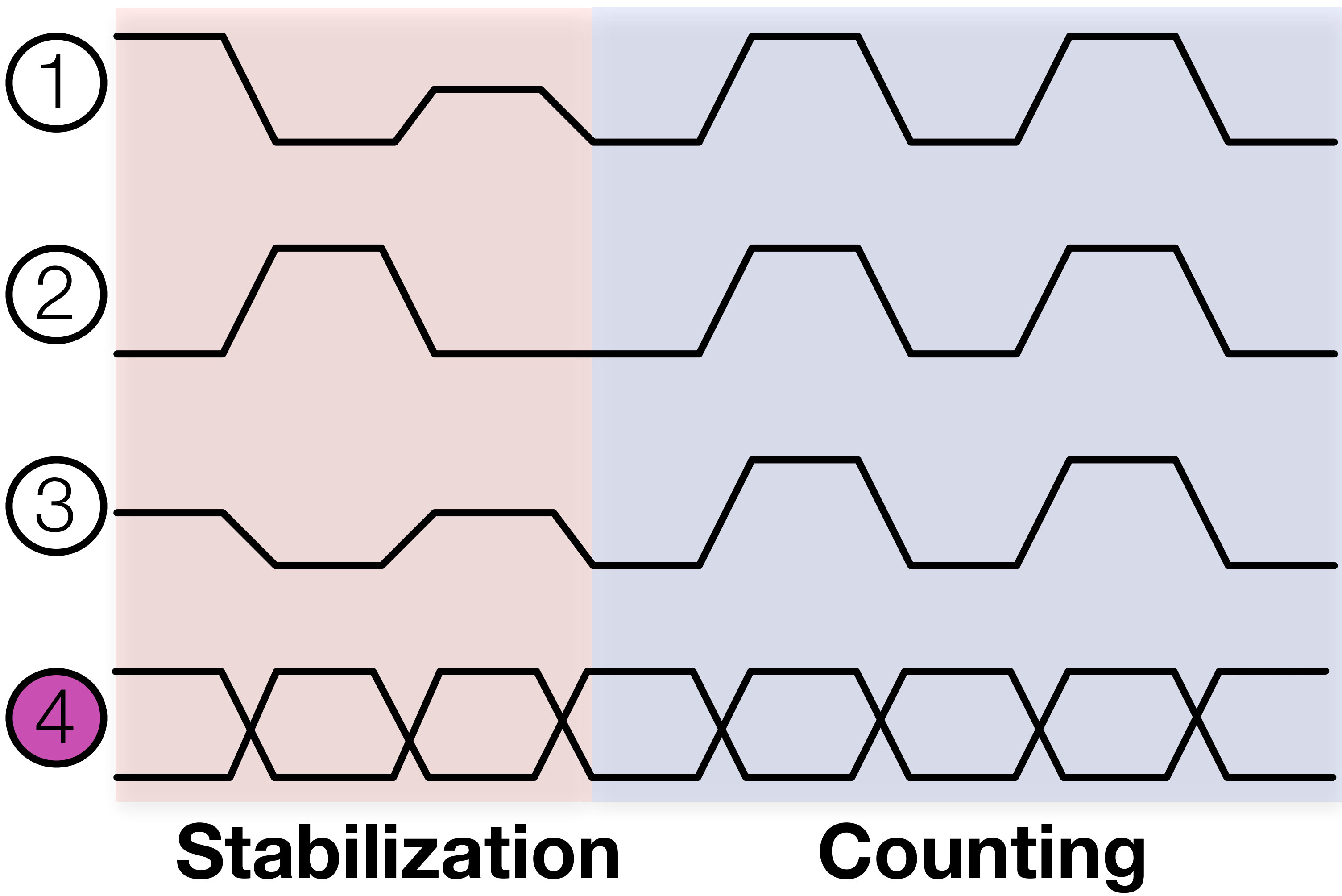
Tolerating Byzantine failures



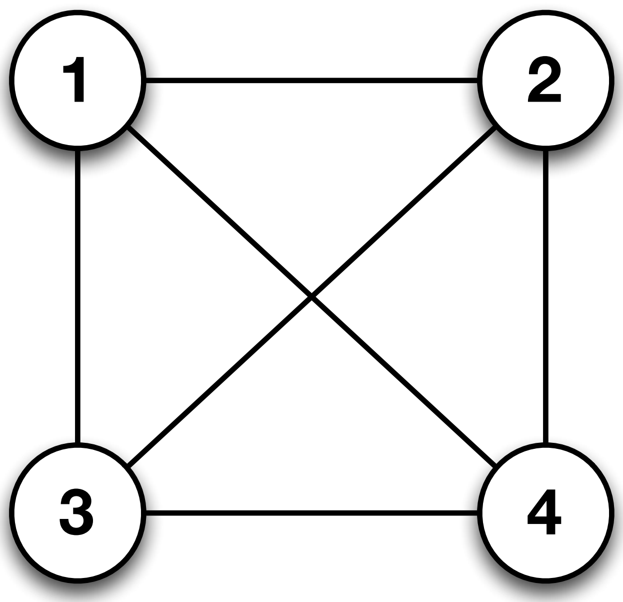
 can send *different* messages to non-faulty nodes!

Note: Counting is easy if self-stabilization is not required (fixed starting state).

Fault-tolerant counting

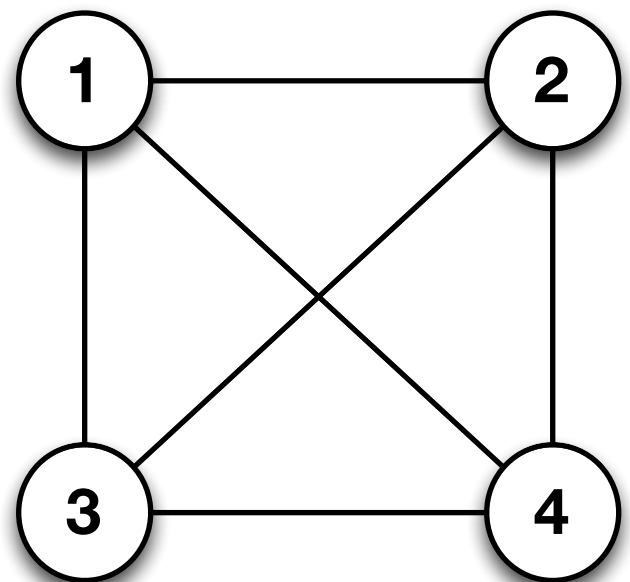


The model with failures

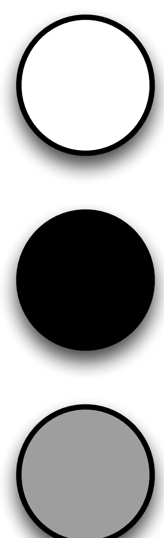


- n processors
- s states
- arbitrary initial state
- at most f Byzantine nodes

An example



- n = 4 processors
- s = 3 states
- f = 1



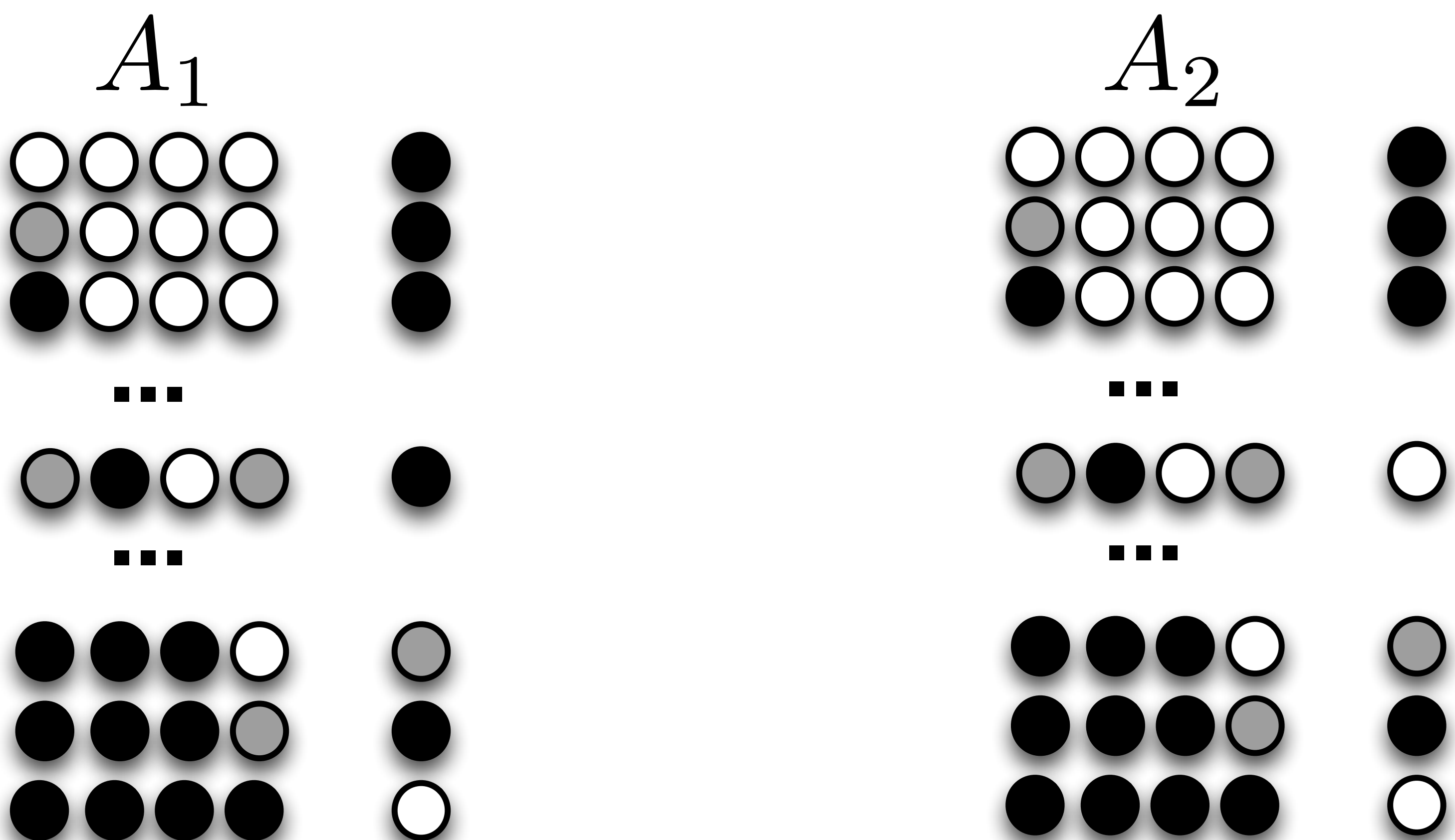
0 = even

1 = odd

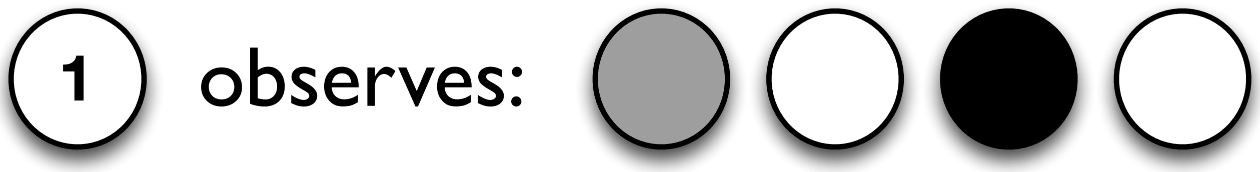
2 = auxiliary state

Algorithms

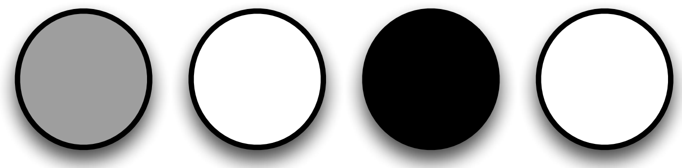
Algorithm **A** gives a transition function for each node *i*:



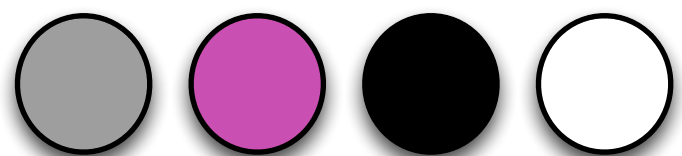
Actual vs observed states



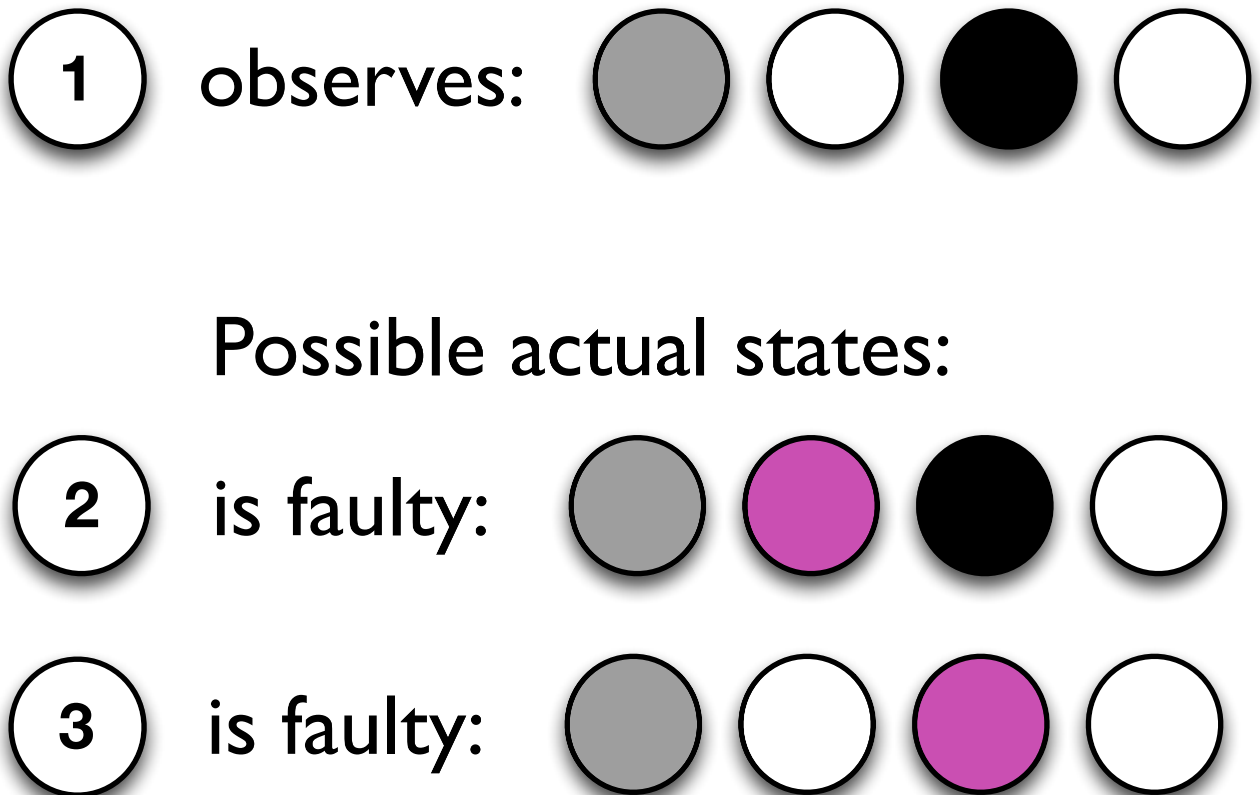
Actual vs observed states

1 observes: 

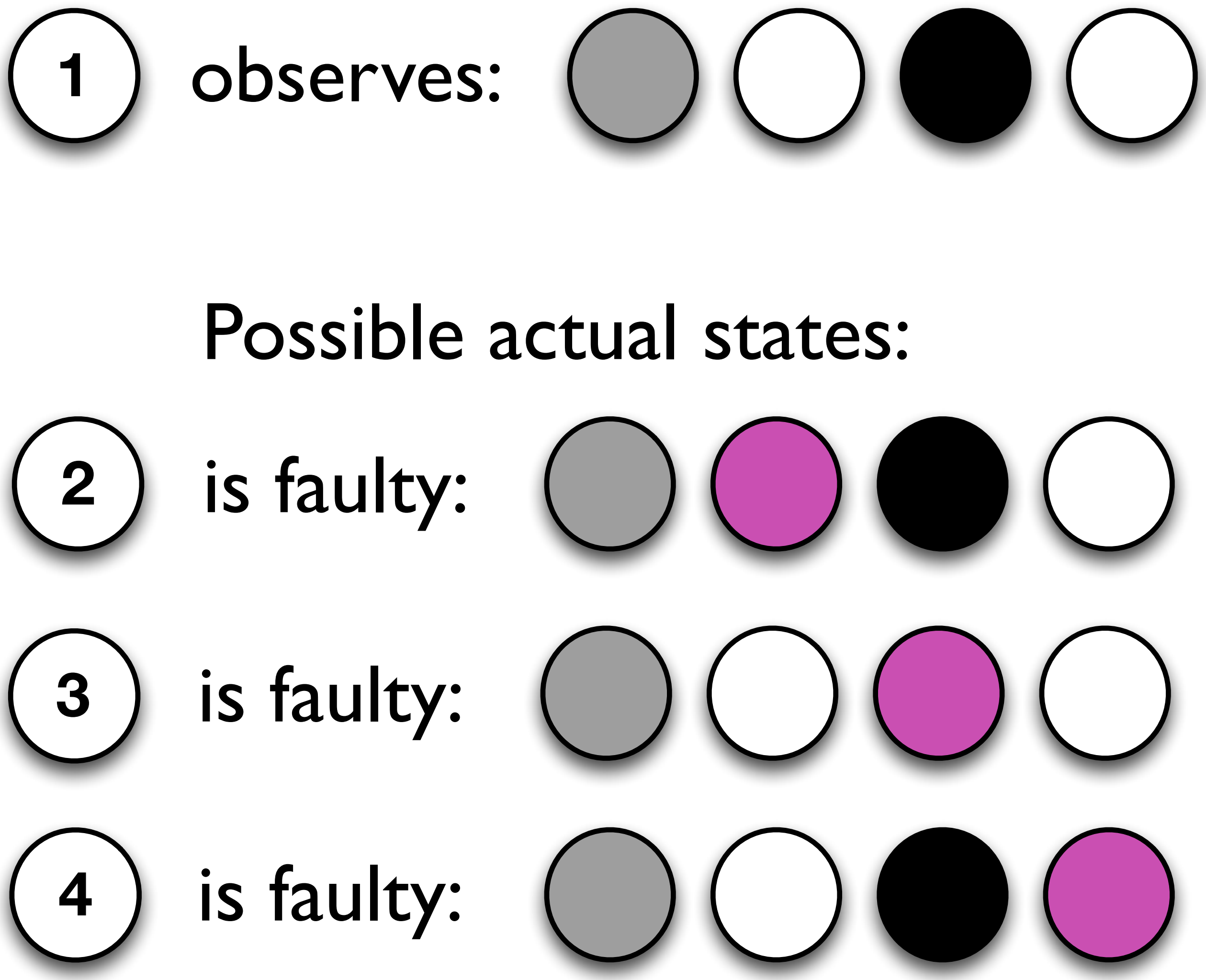
Possible actual states:

2 is faulty: 

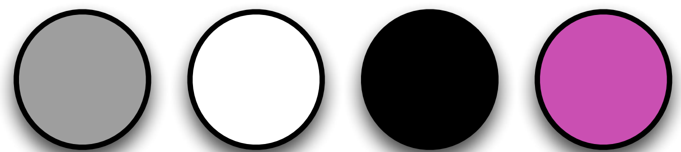
Actual vs observed states

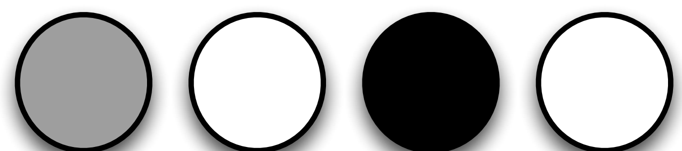


Actual vs observed states

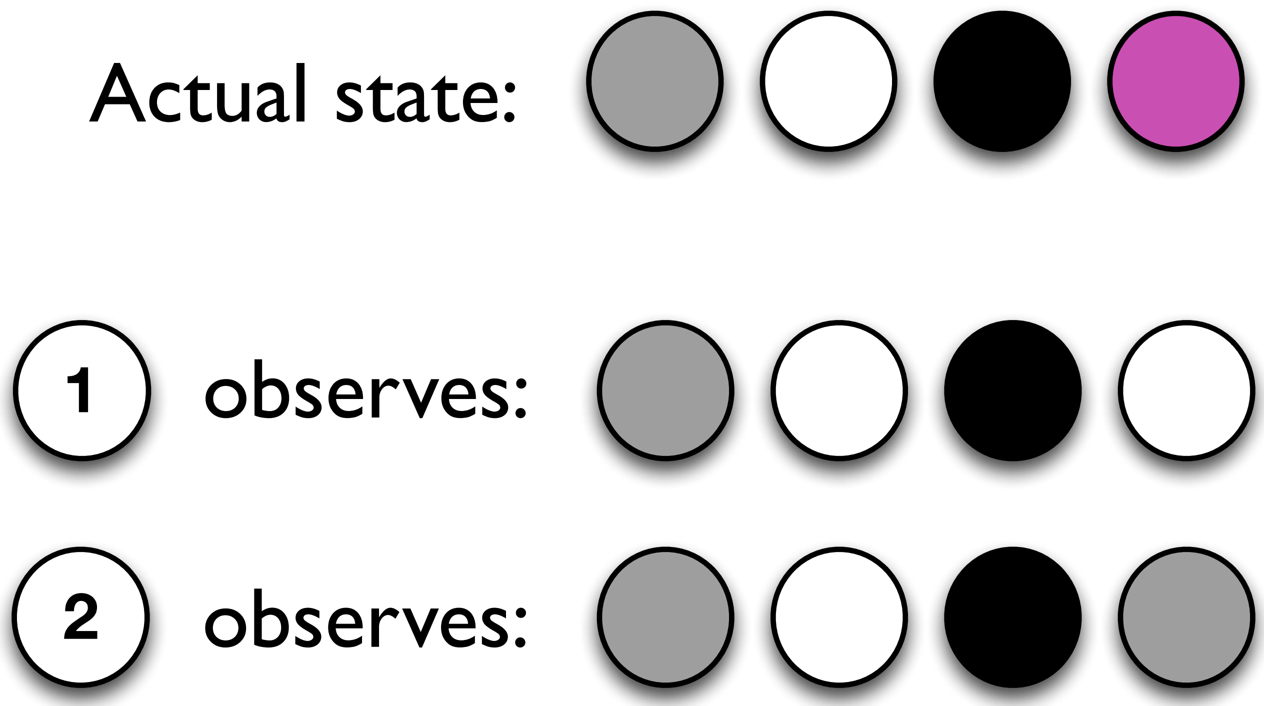


Actual vs observed states

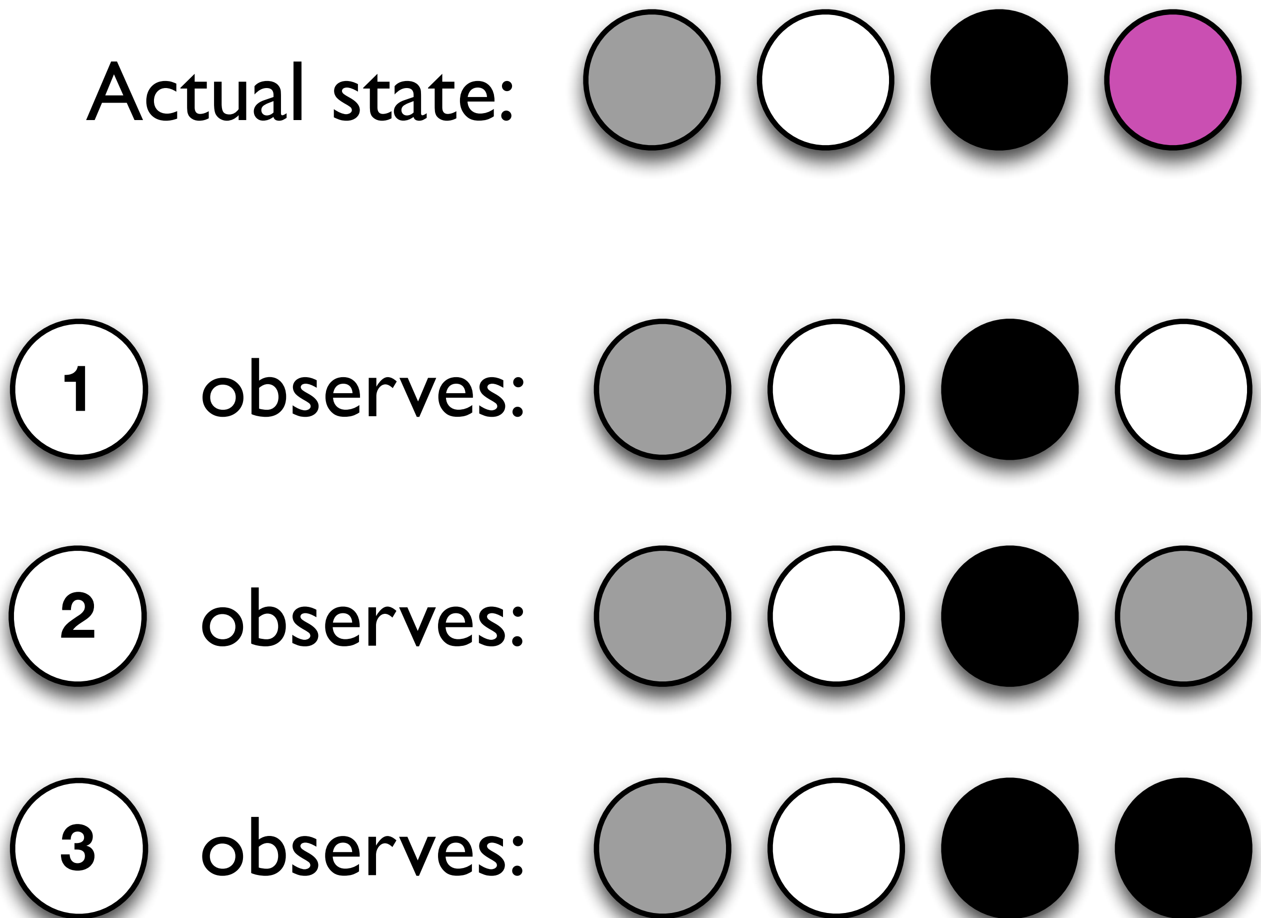
Actual state: 

1 observes: 

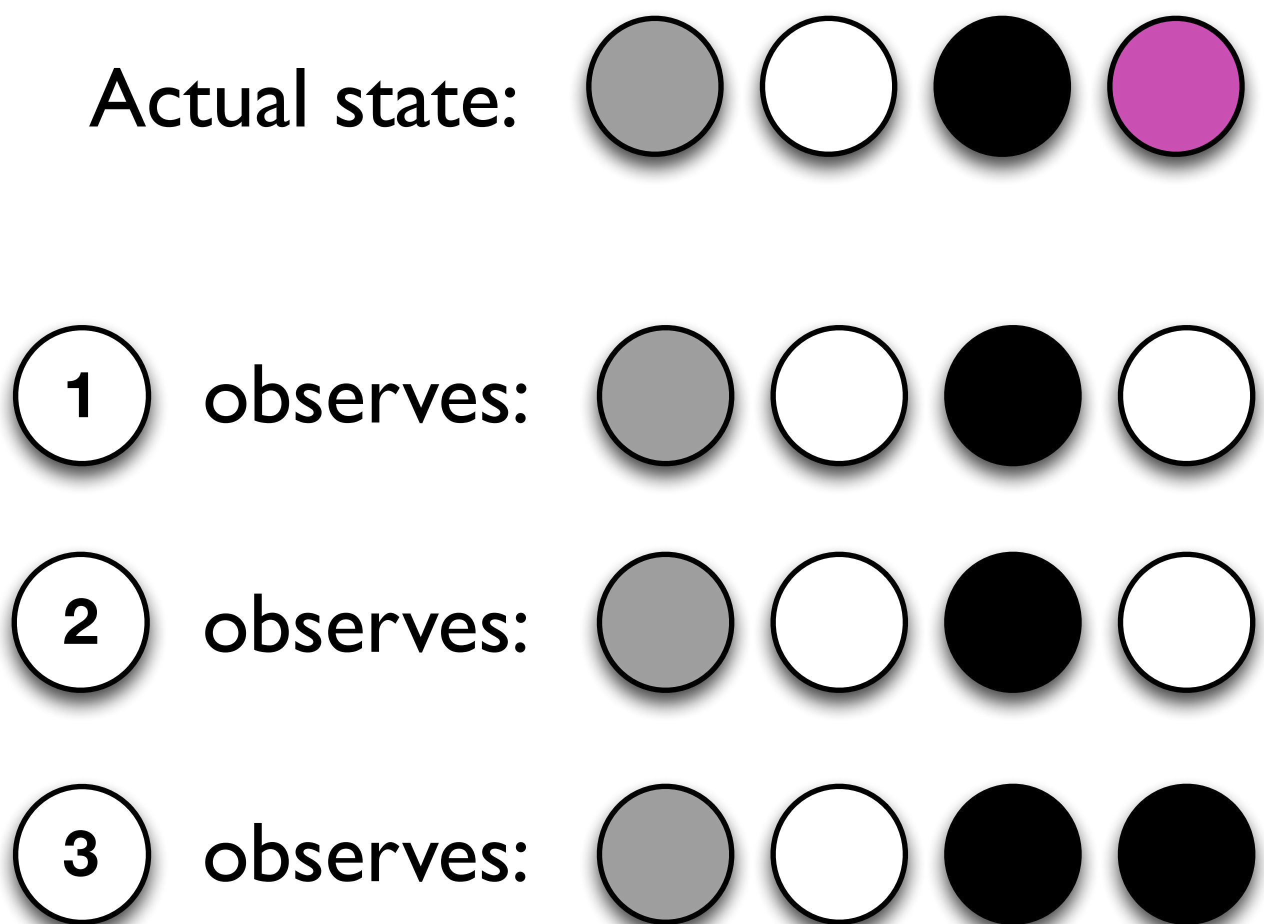
Actual vs observed states



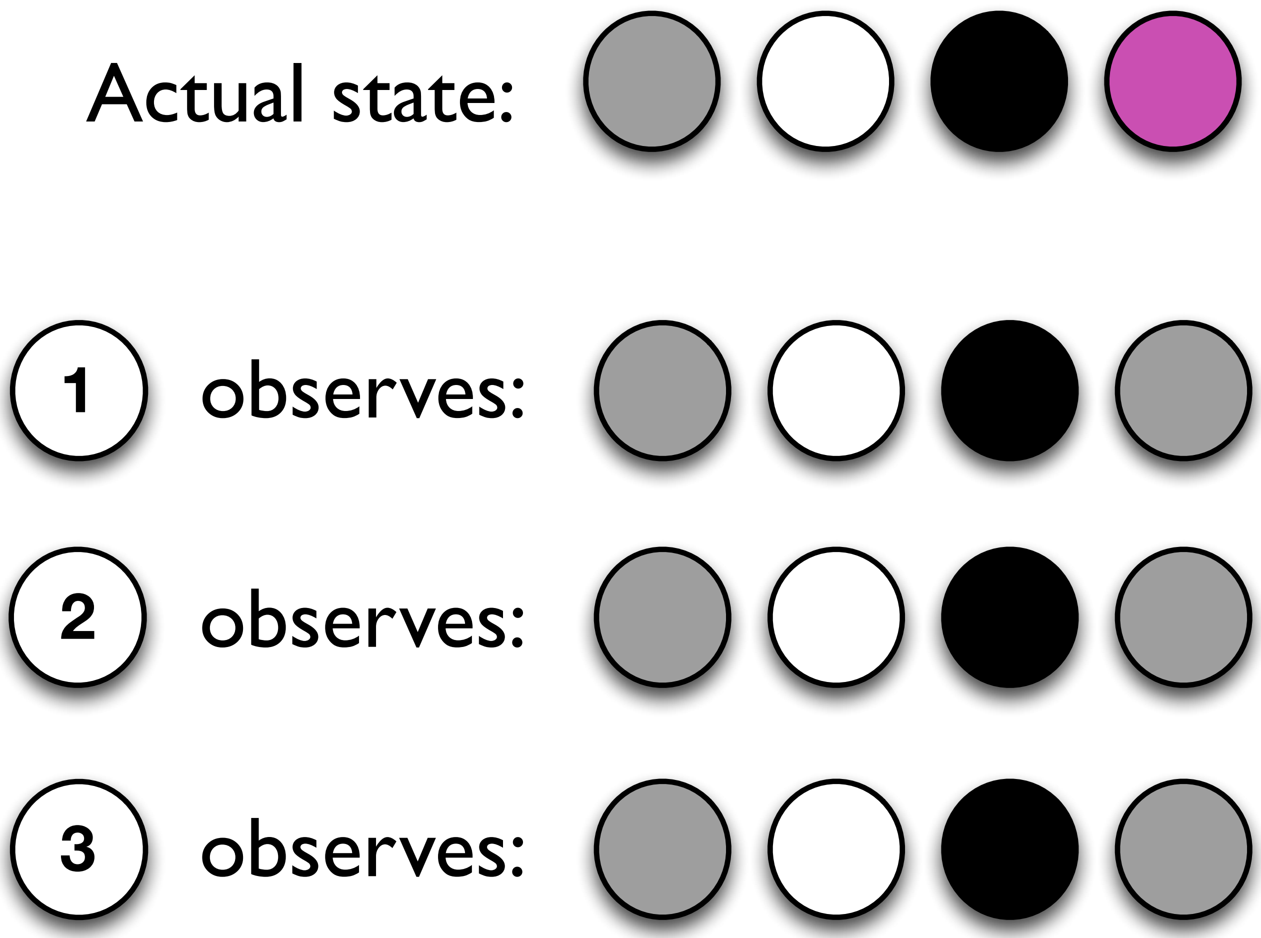
Actual vs observed states



Actual vs observed states



Actual vs observed states



Generalizing from a base case

Suppose we have algorithm **A** that

- solves the counting problem for n nodes
- uses s states per node
- tolerates up to f faulty nodes
- stabilizes in t steps

Generalizing from a base case

Suppose we have algorithm **A** that

- solves the counting problem for n nodes
- uses s states per node
- tolerates up to f faulty nodes
- stabilizes in t steps

$\Rightarrow$ There is an algorithm **B** that solves the counting problem for $n+1$ with the same parameters.

Some basic facts

- How many states do we need?
  - $s \geq 2$
- How many faults can we tolerate?
  - $f < n/3$
- How fast can we stabilize?
  - $t > f$

Prior work

Prior algorithms:

- *deterministic* algorithms with very large *s*
- *randomized* algorithms with small *s*

But are there deterministic algorithms with small *s*?

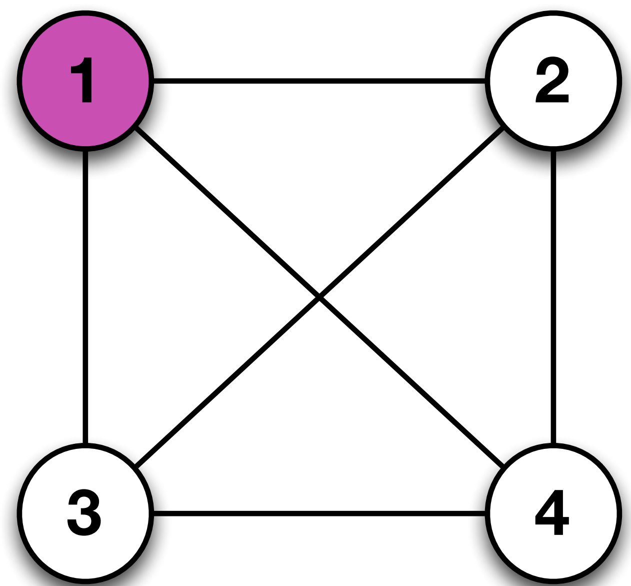
Proving correctness

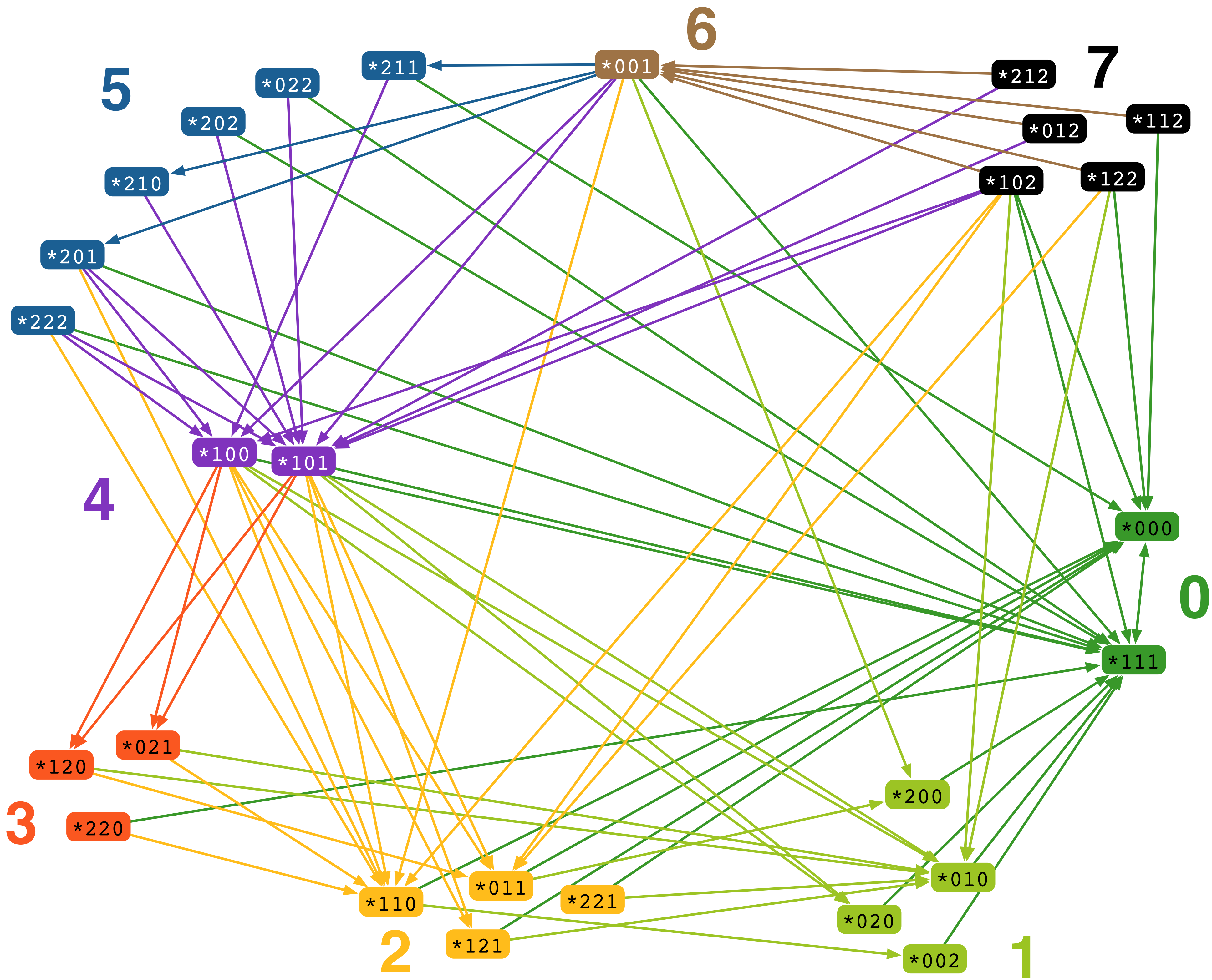
Given algorithm **A**, how to prove it correct?

- Let F be a set of faulty nodes, $|F| \leq f$
- Construct a projection graph G_F from **A**
- Nodes = actual states
- Edges = possible state transitions

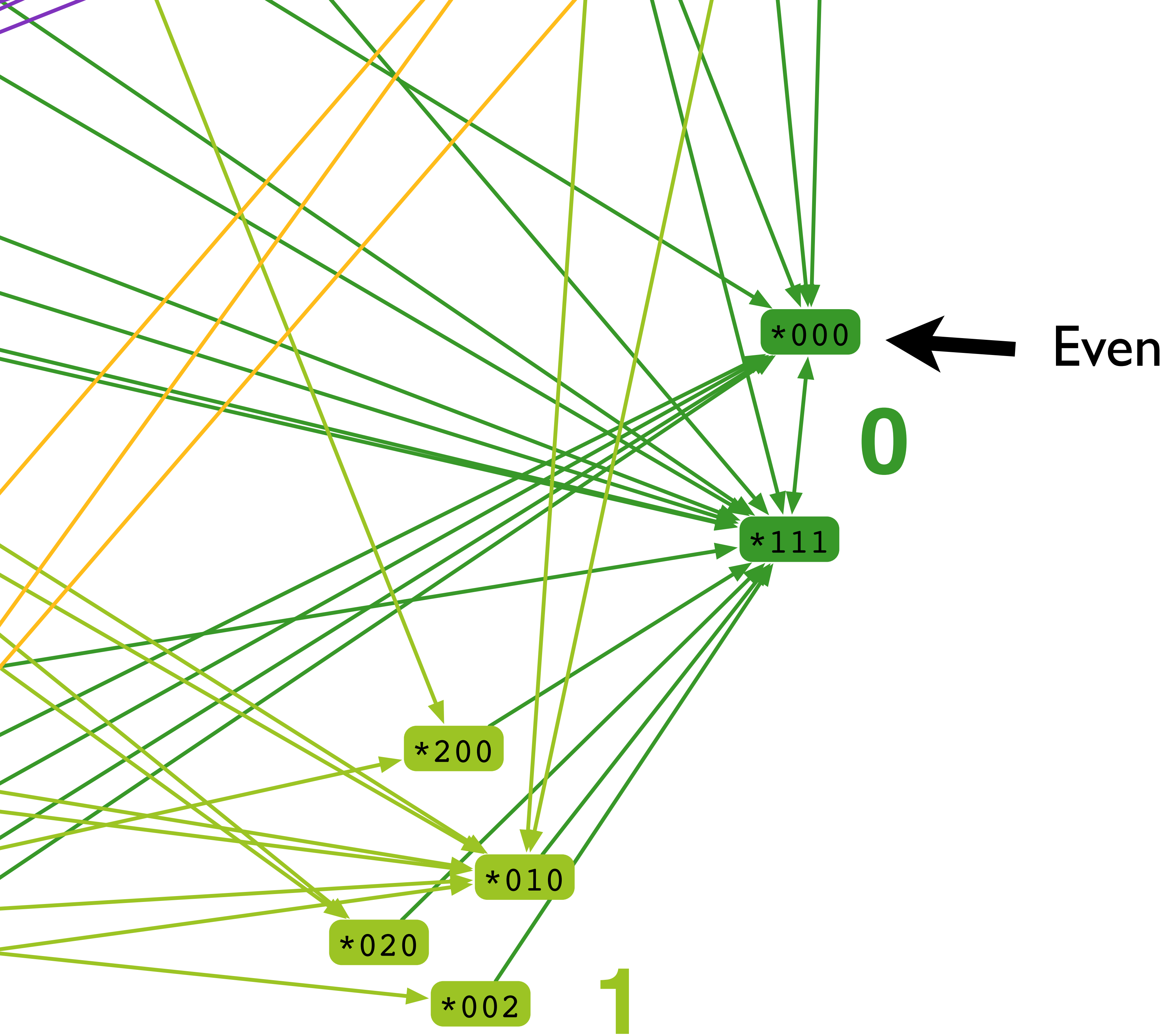
An example

- $n = 4$
- $s = 3$
- $F = \{1\}$

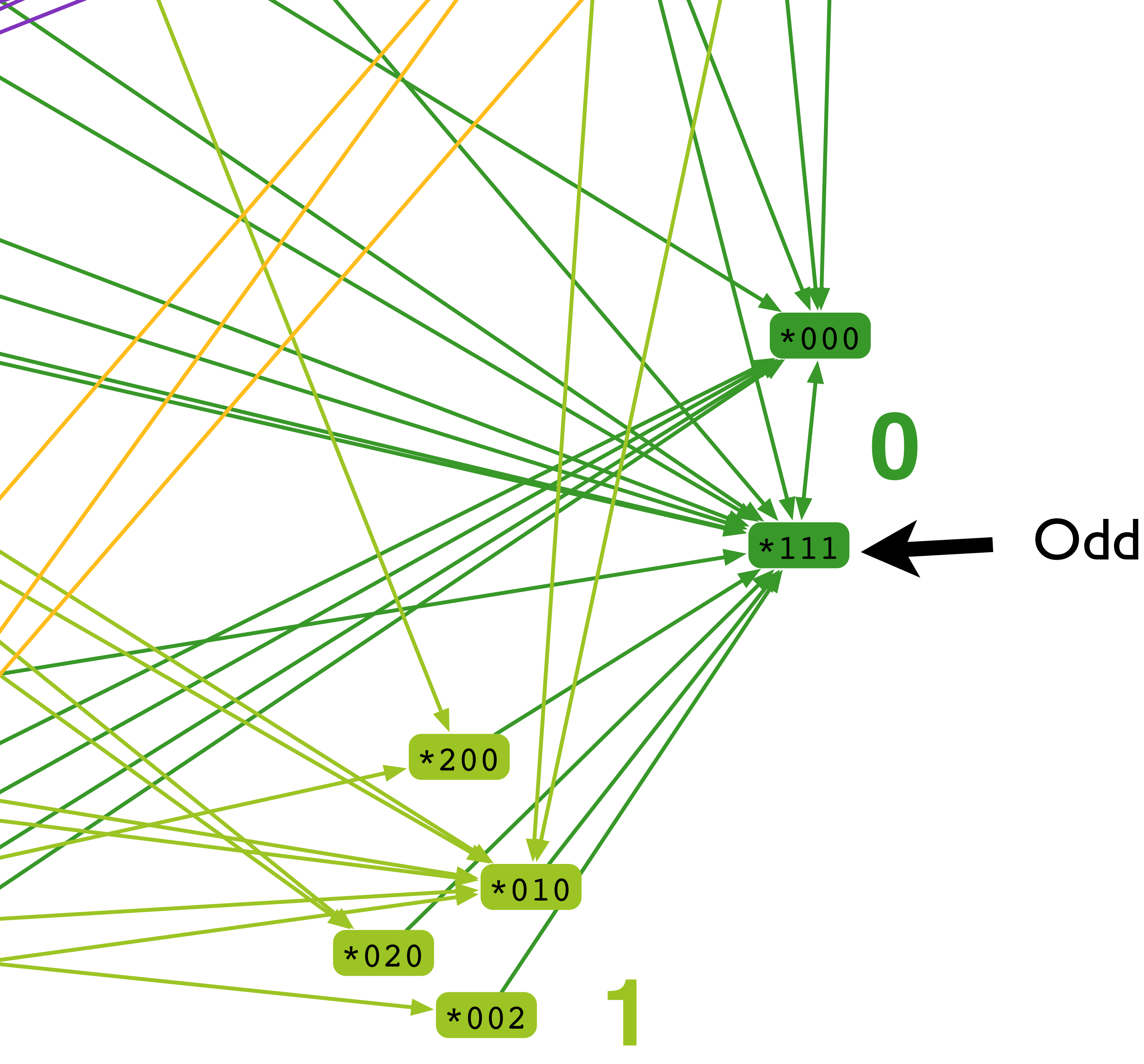




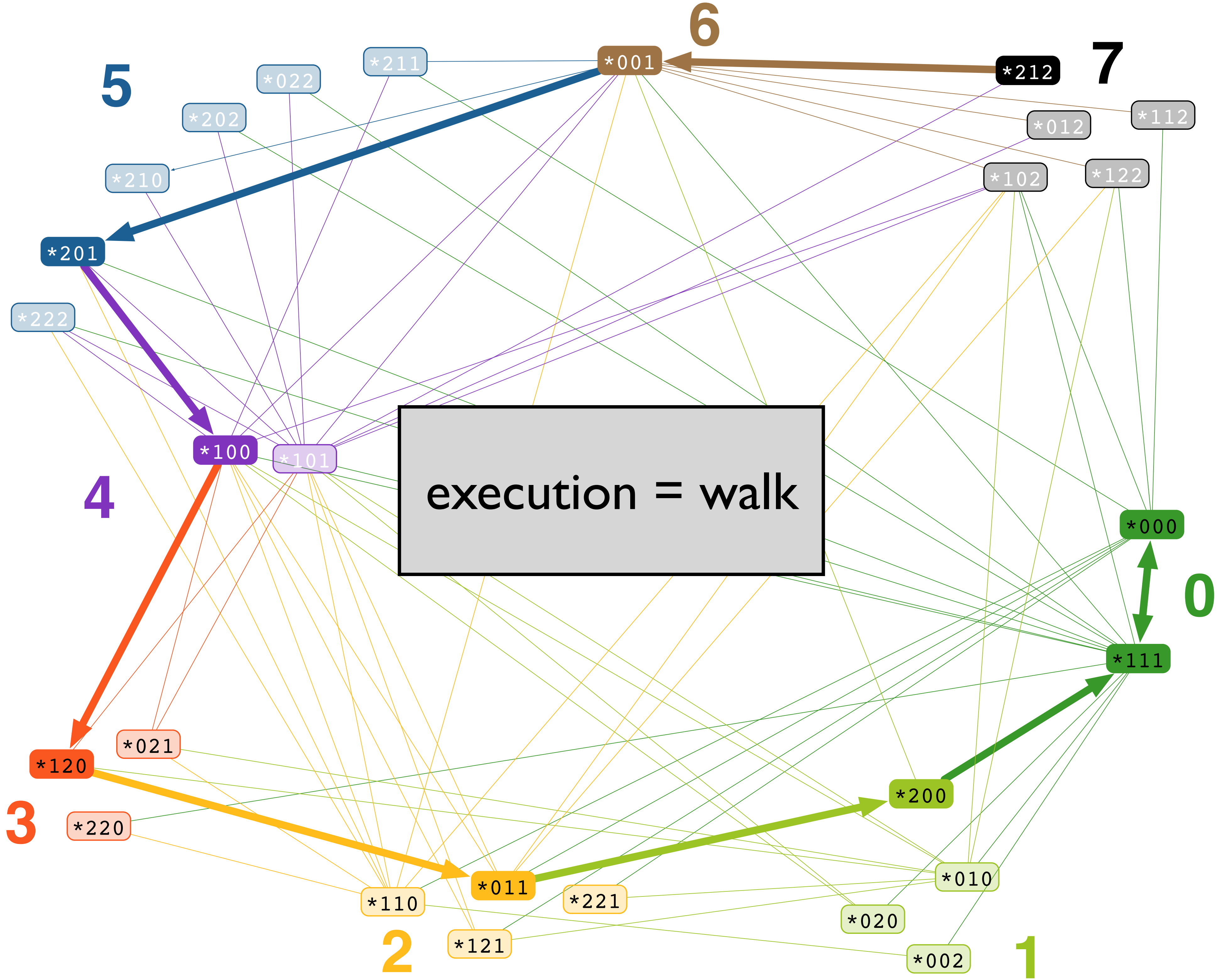
6
*001
7
*212
*112
*012
*102
*122
5
*211
*022
*202
*210
*201
*222
4
*100
*101
0
*000
*111
3
*120
*021
*220
2
*110
*011
*221
*121
1
*200
*010
*020
*002



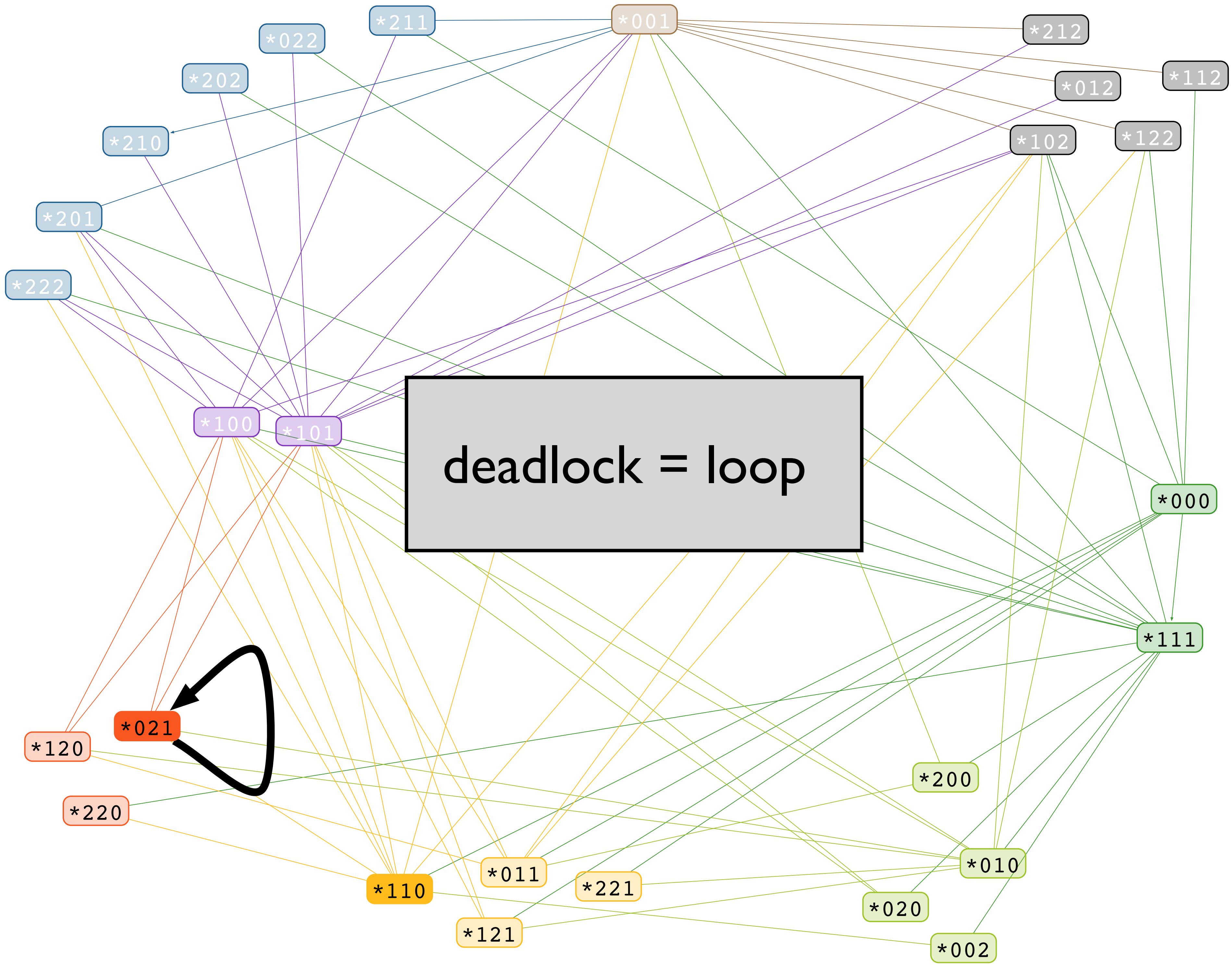
*000
Even
0
*111
*200
*010
*020
*002
1


*000
0
*111
Odd
*200
*010
*020
*002
1

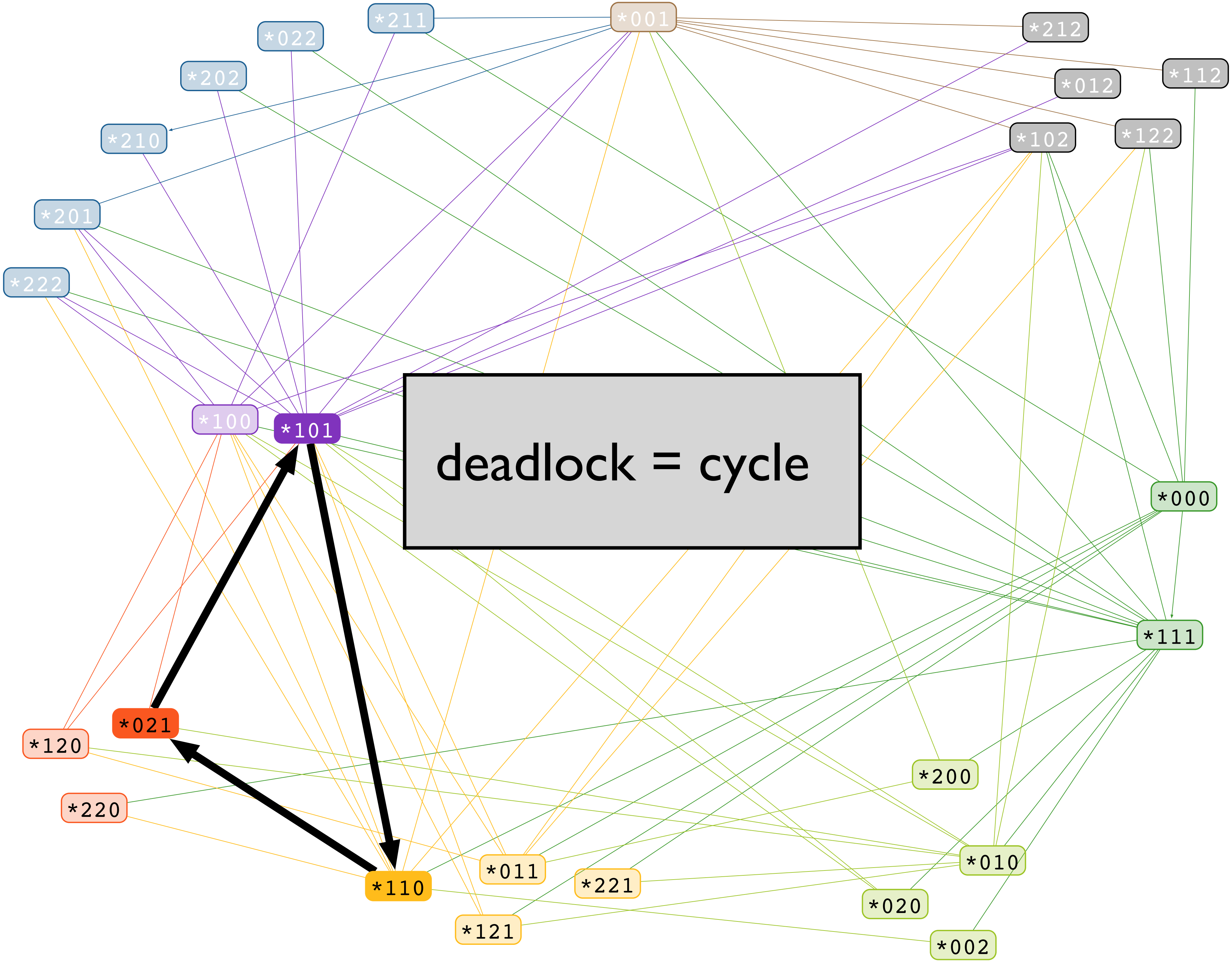
Easy to talk about graph properties!



execution = walk
0
1
2
3
4
5
6
7
*000
*111
*200
*011
*120
*100
*201
*001
*212
*010
*020
*002
*110
*121
*221
*021
*220
*101
*222
*210
*202
*022
*211
*012
*112
*102
*122



*001
*211
*022
*202
*210
*201
*222
*212
*112
*012
*102
*122
*100
*101
deadlock = loop
*000
*111
*021
*120
*220
*200
*010
*011
*221
*110
*020
*121
*002


deadlock = cycle
*001
*211
*022
*202
*210
*201
*222
*212
*112
*012
*102
*122
*100
*101
*000
*111
*021
*120
*220
*200
*110
*011
*221
*010
*020
*121
*002

Proving correctness

Algorithm **A** is correct iff for all F the graph G_F satisfies

1. G_F is loopless $\Leftrightarrow$ no deadlocks

Proving correctness

Algorithm **A** is correct iff for all F the graph G_F satisfies

1. G_F is loopless	$\Leftrightarrow$	no deadlocks
2. All nodes have a path to **0** and **1**	$\Leftrightarrow$	stabilization

Proving correctness

Algorithm **A** is correct iff for all F the graph G_F satisfies

1. G_F is loopless	$\Leftrightarrow$	no deadlocks
2. All nodes have a path to **0** and **1**	$\Leftrightarrow$	stabilization
3. $\{\mathbf{0},\mathbf{1}\}$ is the only cycle	$\Leftrightarrow$	counting

Finding an algorithm

The size of the search space is s^b where $b = ns^n$.

Finding an algorithm

The size of the search space is s^b where $b = ns^n$.

parameters	search space
$n = 4$ $s = 2$	$2^{64} \approx 10^{19}$

Finding an algorithm

The size of the search space is s^b where $b = ns^n$.

parameters	search space
n = 4 s = 2	$2^{64} \approx 10^{19}$
n = 4 s = 3	$3^{324} \approx 10^{154}$

SAT solving

Propositional satisfiability

Problem: Given a propositional formula Ψ, does there exist a satisfying variable assignment?

Propositional satisfiability

Problem: Given a propositional formula Ψ, does there exist a satisfying variable assignment?

Example 1: $(x_1 \vee \neg x_2) \wedge (x_1 \rightarrow x_2)$ **SAT**

$$\begin{cases} x_1 = 0 \\ x_2 = 0 \end{cases}$$

Propositional satisfiability

Problem: Given a propositional formula Ψ, does there exist a satisfying variable assignment?

Example 2: $x_1 \wedge \neg x_2 \wedge (x_1 \rightarrow x_2)$ **UNSAT**

SAT solvers

- Fast in practice
- New solvers and techniques are developed all the time
- Input in *conjunctive normal form* (CNF):

$$\Psi = \bigwedge C_i$$

$$C_i = \ell_{i_1} \vee \cdots \vee \ell_{i_k}$$

Proving correctness (revisited)

Algorithm **A** is correct iff for all F the graph G_F satisfies

1. G_F is loopless	$\Leftrightarrow$	no deadlocks
2. All nodes have a path to **0** and **1**	$\Leftrightarrow$	stabilization
3. $\{\mathbf{0},\mathbf{1}\}$ is the only cycle	$\Leftrightarrow$	counting

Searching algorithms using SAT

Introduce the following variables:

$x_{i,u,s}$ corresponds to $A_i(u) = s$

Searching algorithms using SAT

Introduce the following variables:

$x_{i,u,s}$ corresponds to $A_i(u) = s$

$e_{q,r}$ denotes the presence of an edge (q, r) in a projection graph

Searching algorithms using SAT

Introduce the following variables:

$x_{i,u,s}$ corresponds to $A_i(u) = s$

$e_{q,r}$ denotes the presence of an edge (q, r) in a projection graph

$p_{q,r}$ denotes a path $q \rightsquigarrow r$

Searching algorithms using SAT

Introduce the following variables:

$x_{i,u,s}$ corresponds to $A_i(u) = s$

$e_{q,r}$ denotes the presence of an edge (q, r) in a projection graph

$p_{q,r}$ denotes a path $q \rightsquigarrow r$

$$x_{i,u,s} \Rightarrow e_{q,r} \Rightarrow p_{q,r}$$

Reminder: Proving correctness

Algorithm **A** is correct iff for all F the graph G_F satisfies

1. G_F is loopless	$\Leftrightarrow$	no deadlocks
2. All nodes have a path to **0** and **1**	$\Leftrightarrow$	stabilization
3. $\{\mathbf{0},\mathbf{1}\}$ is the only cycle	$\Leftrightarrow$	counting

Reminder: Proving correctness

Algorithm **A** is correct iff for all *F* the graph G_F satisfies

1. G_F is loopless $\Leftrightarrow \neg e_{q,q}$

2. All nodes have a path to **0** and **1** $\Leftrightarrow p_{q,\mathbf{0}}$

3. {**0**,**1**} is the only cycle $\Leftrightarrow \neg p_{q,q}$ if $q \notin \{\mathbf{0}, \mathbf{1}\}$

   $e_{\mathbf{0},\mathbf{1}} \wedge e_{\mathbf{1},\mathbf{0}}$

Ex.: 4 nodes, 3 states, 1 faulty

```
p cnf 6120 157900
1 2 3 0
 ....
745 -176 -218 -227 0
 ....
5522 -5860 -5513 0
 ....
-3204 0
```

Ex.: 4 nodes, 3 states, 1 faulty

vars clauses

```
p cnf 6120 157900
1 2 3 0
 ....
745 -176 -218 -227 0
 ....
5522 -5860 -5513 0
 ....
-3204 0
```

Ex.: 4 nodes, 3 states, 1 faulty

```
p cnf 612
1 2 3 0
 ....
745 -176 -218 -227 0
 ....
5522 -5860 -5513 0
 ....
-3204 0
```

$$\boxed{x_{0,(0201),0} \vee x_{0,(0201),1} \vee x_{0,(0201),2}}$$

Ex.: 4 nodes, 3 states, 1 faulty

$$(x_{0,1001,1} \wedge x_{2,1101,1} \wedge x_{3,1001,1}) \rightarrow e_{*110,*111}$$

```
p c
1 2
 ....
745 -176 -218 -227 0
 ....
5522 -5860 -5513 0
 ....
-3204 0
```

Ex.: 4 nodes, 3 states, 1 faulty

```
p cnf 6120 157900
1 2 3 0
 ..
745
```

$$(p_{111*,000*} \wedge p_{202*,111*}) \rightarrow p_{202*,000*}$$

```
 ....
5522 -5860 -5513 0
 ....
-3204 0
```

Ex.: 4 nodes, 3 states, 1 faulty

```
p cnf 6120 157900
1 2 3 0
 ....
745 -176 -218 -227 0
 ....
5522 -5860 -5513 0
 ....
-3204 0
```

$$\boxed{\neg e_{000*,000*}}$$

Ex.: 4 nodes, 3 states, 1 faulty

- 6120 variables and 15790 clauses
- $2^{6120} \approx 10^{1842}$ possible assignments
- plingeling solves the instance in less than 2 seconds

Main results, $f = 1$

If $4 \leq n \leq 5$:

- no 2-state algorithm
- ..but 3 states suffice

Main results, $f = 1$

If $4 \leq n \leq 5$:

- no 2-state algorithm
- ..but 3 states suffice

If $n \geq 6$:

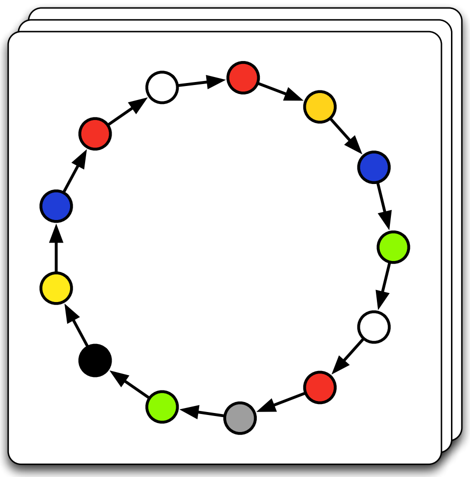
- 2 states always suffice
- ..but increasing the number of states seems to yield faster algorithms

What next?

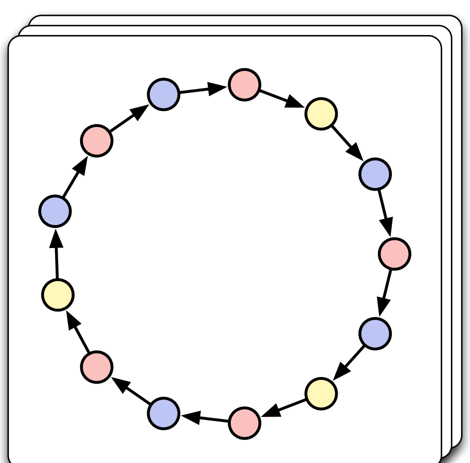
- What about $f = 2$?
- Instances are very large; no luck so far
- An inductive approach for f as well?

Graph coloring and max cut

Distributed graph coloring

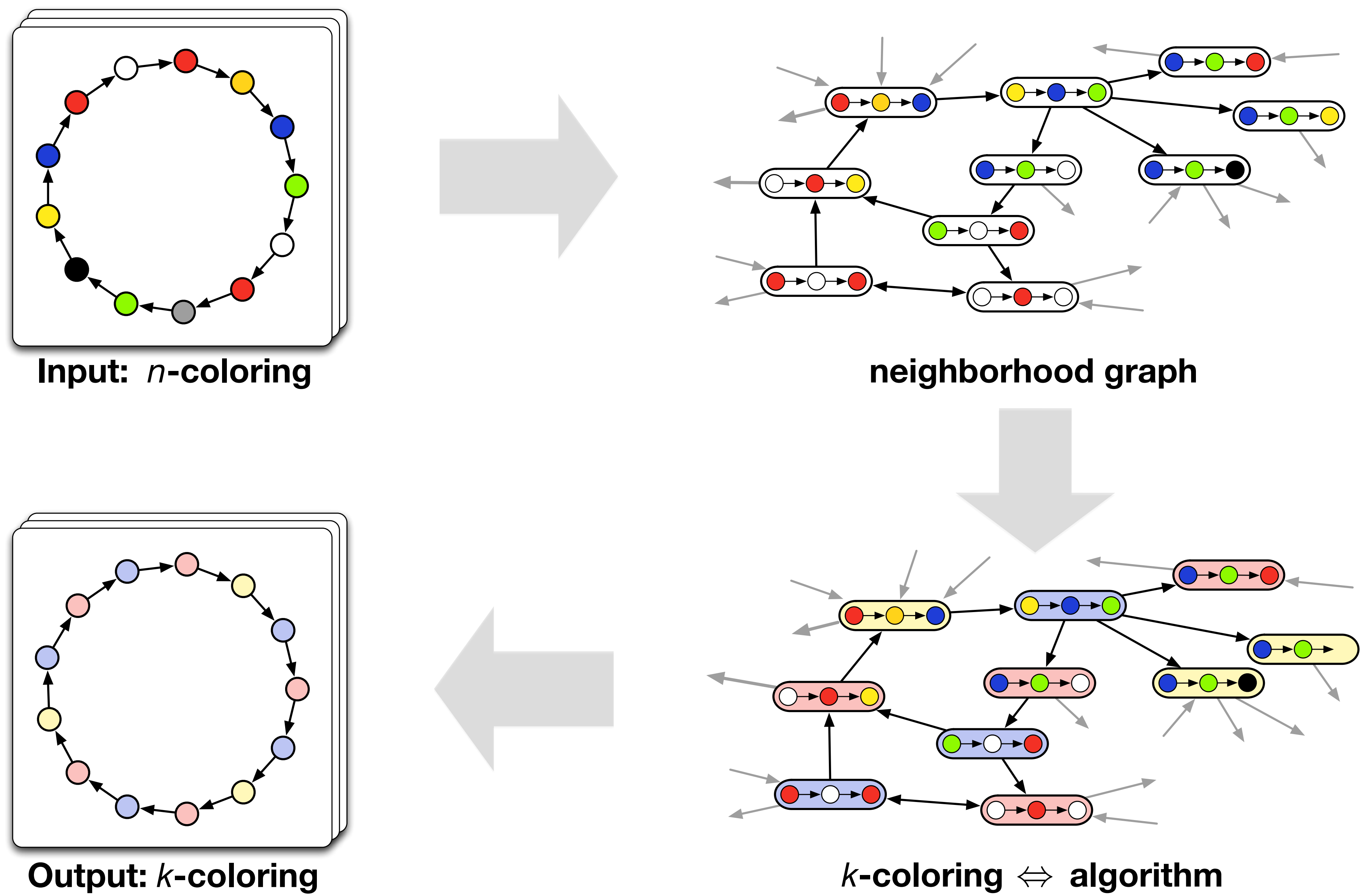


Input: ***n*-coloring**

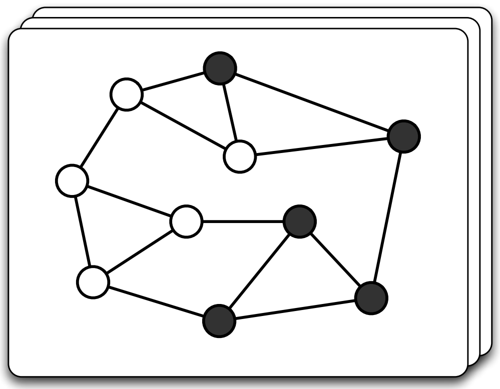


Output: ***k*-coloring**

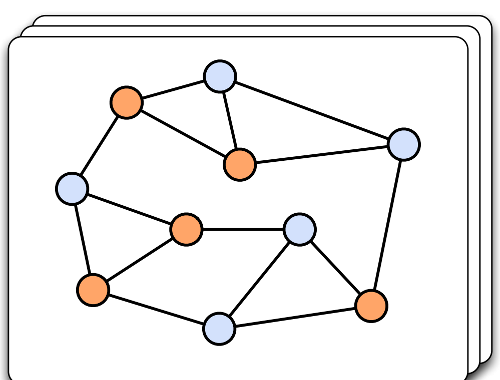
Distributed graph coloring



Randomized max cut

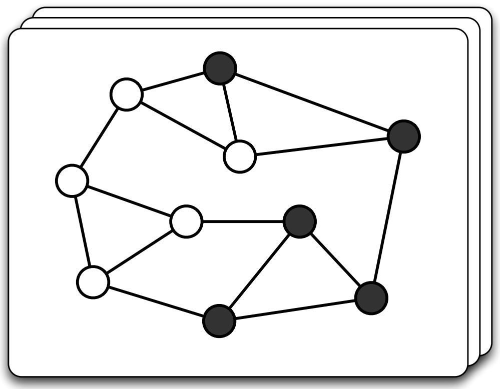


Input: random cut

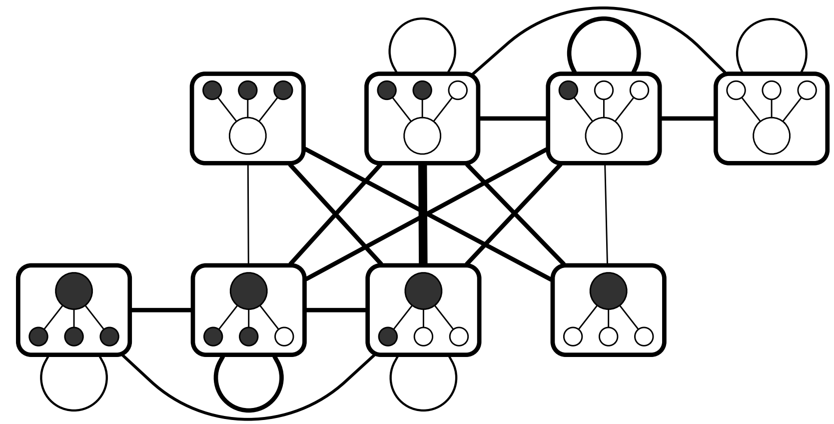


Output: better cut

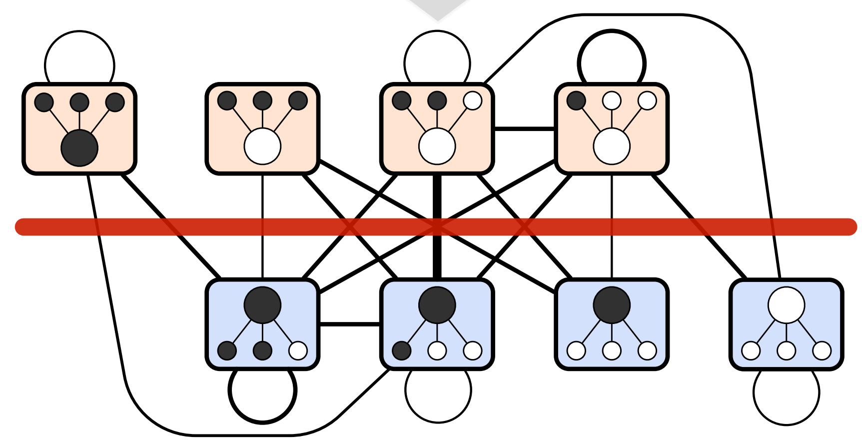
Randomized max cut



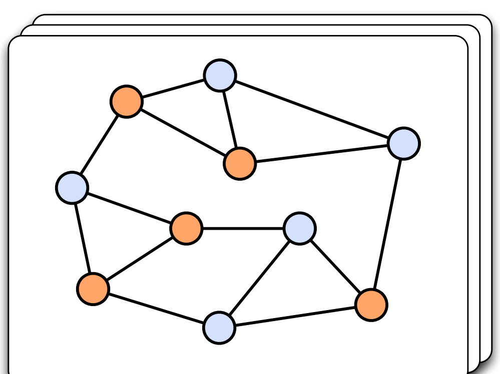
Input: random cut



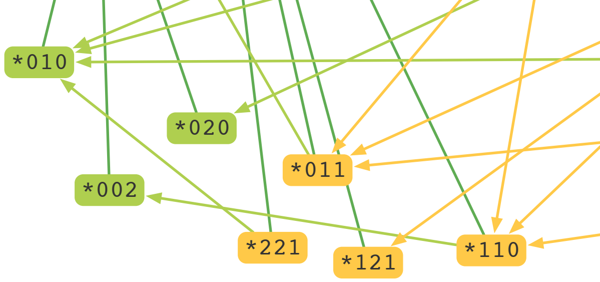
neighborhood graph



cut ⇔ algorithm



Output: better cut



Thanks for listening!