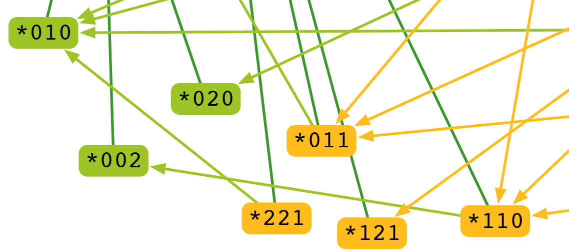
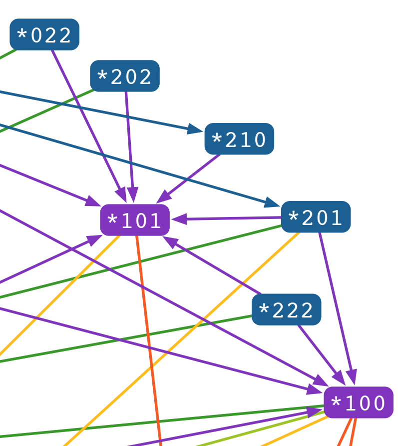


Synchronous counting and computational algorithm design

Danny Dolev
Hebrew University of Jerusalem

Christoph Lenzen
MIT

Janne H. Korhonen **Joel Rybicki** **Jukka Suomela**
University of Helsinki & HIIT



November 16, 2013
SSS 2013, Osaka, Japan

What is this talk about?

Developing *compact* fault-tolerant algorithms for a consensus-like problem using *computational techniques*.

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**?"*

Verification vs synthesis

Verification:

"***Check*** *that given* ***A*** *satisfies the specification* ***S****.*"

Synthesis:

"***Construct*** *an* ***A*** *that satisfies a specification* ***S****.*"

Searching for algorithms

How to do a computer search?

Intuitively, the task seems very difficult.

An inductive approach

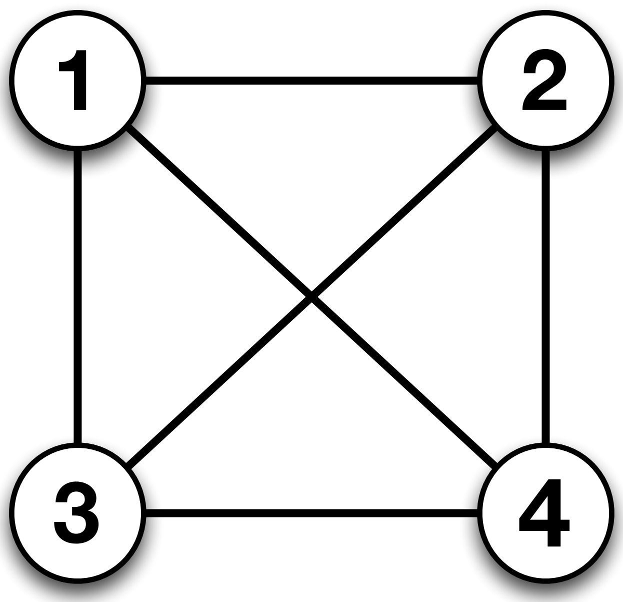
1. Solve a difficult *base case* using computers

2. Construct a *general* solution using the base case

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

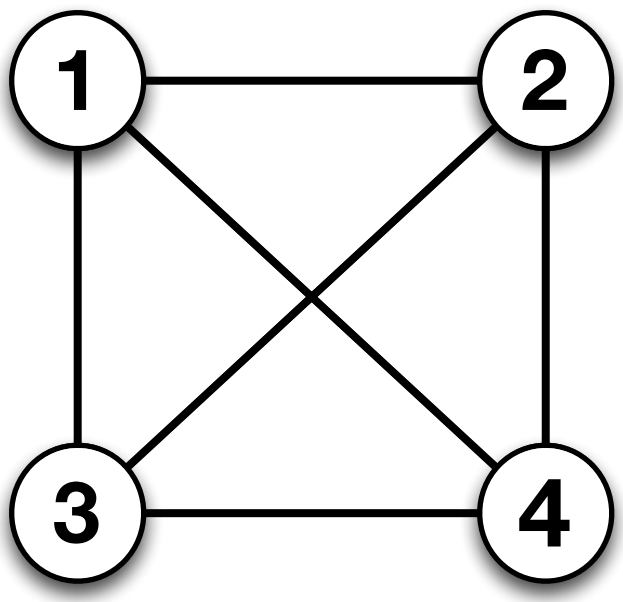
Synchronous counting

The model



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

The model



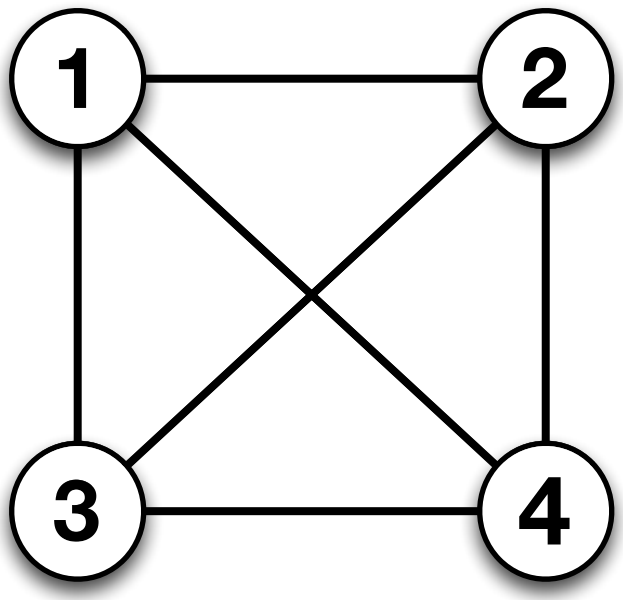
- *n* processors
- *s* states
- arbitrary initial state

Synchronous step:

1. send state to all neighbors

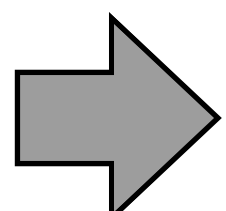
2. update state

The model



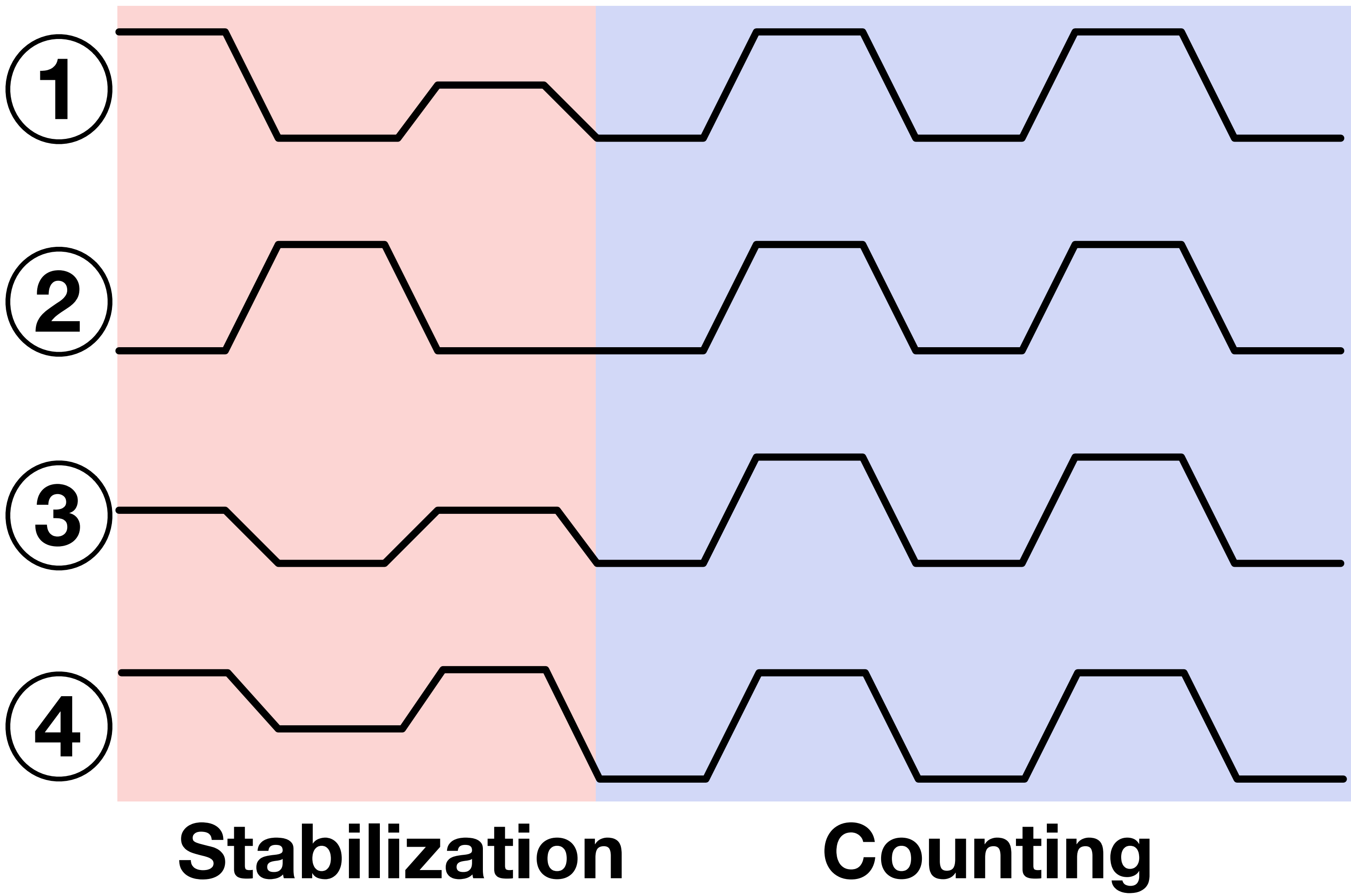
- *n* processors
- *s* states
- arbitrary initial state

Synchronous step:
1. send state to all neighbors
2. update state



algorithm
=
transition function

Self-stabilizing counting

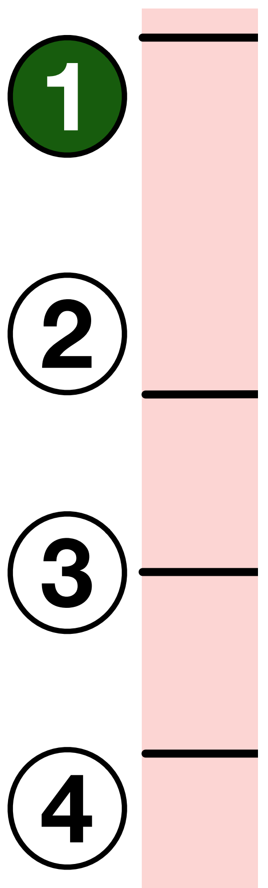


Self-stabilizing counting

A simple algorithm solves the problem

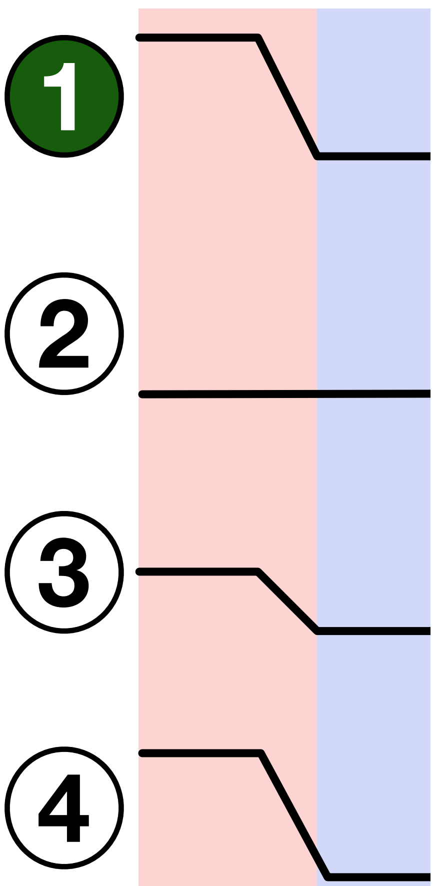
Self-stabilizing counting

Solution: Follow the leader.



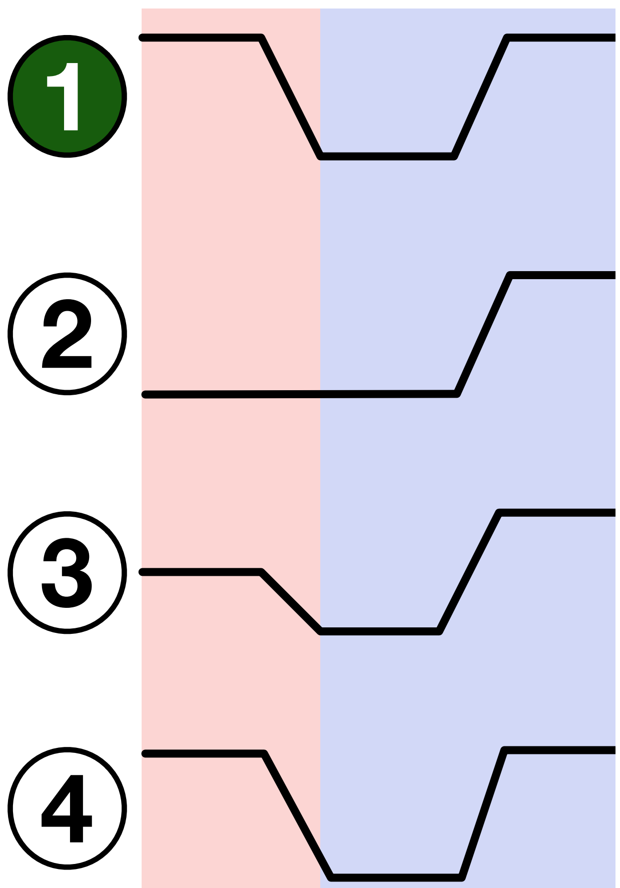
Self-stabilizing counting

Solution: Follow the leader.



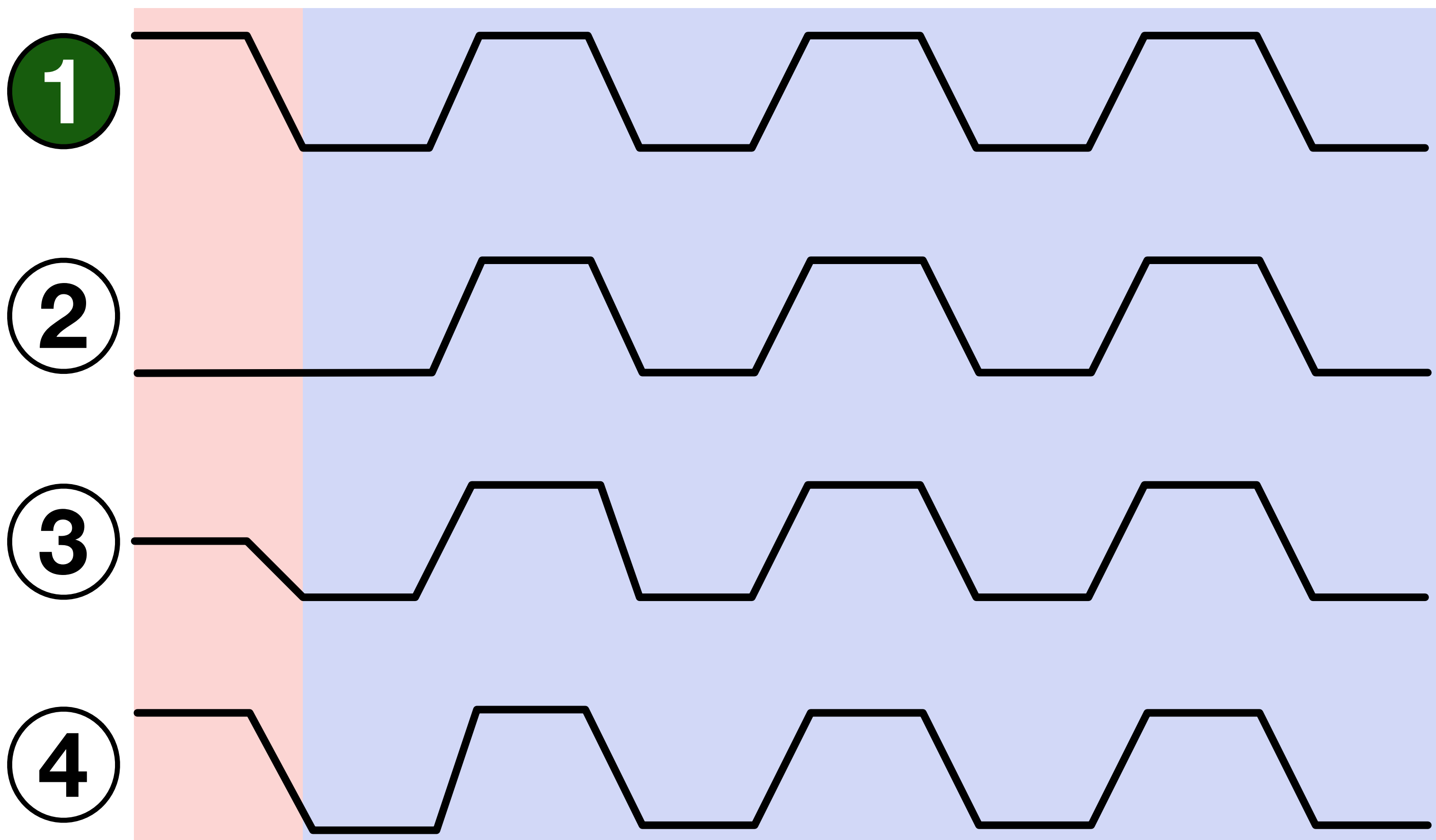
Self-stabilizing counting

Solution: Follow the leader.

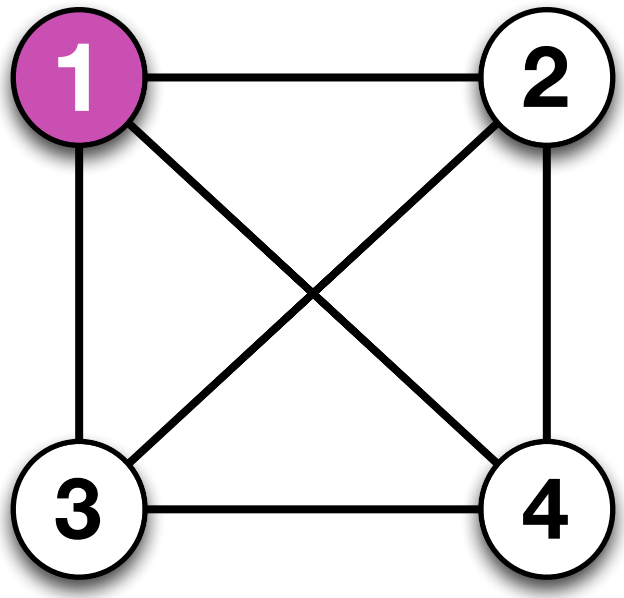


Self-stabilizing counting

Solution: Follow the leader.

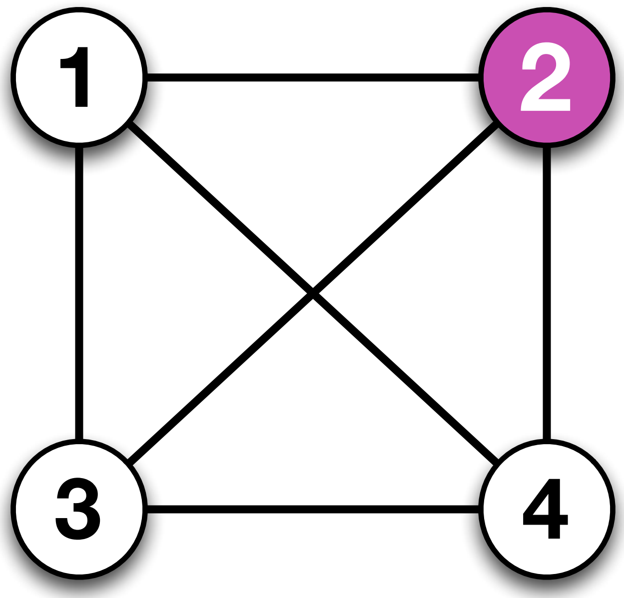


Tolerating Byzantine failures



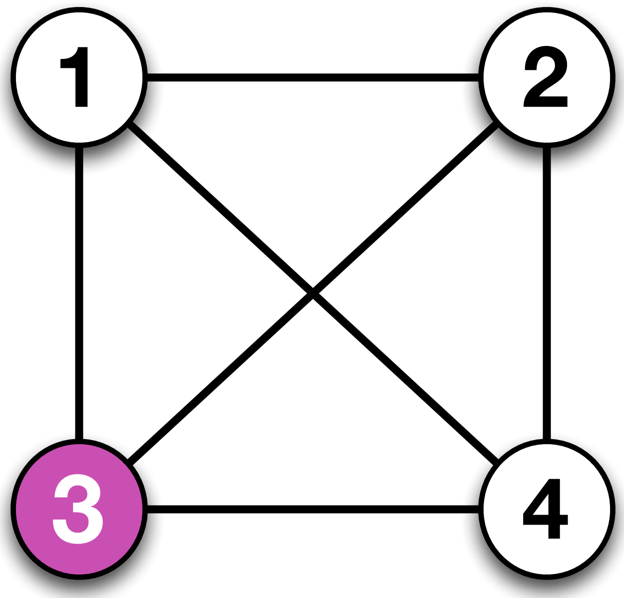
Assume that at most *f* nodes may be **Byzantine**.

Tolerating Byzantine failures



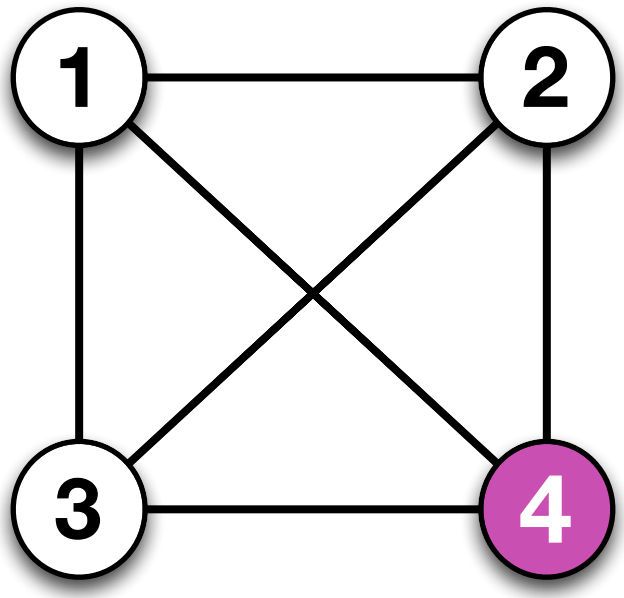
Assume that at most *f* nodes may be **Byzantine**.

Tolerating Byzantine failures



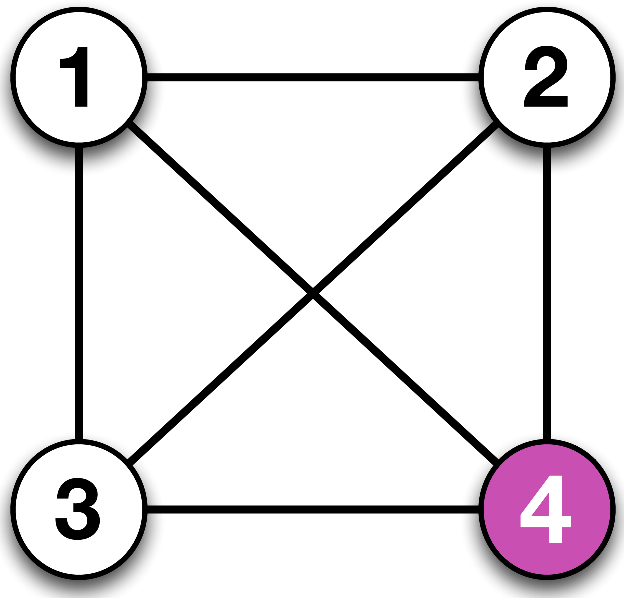
Assume that at most *f* nodes may be **Byzantine**.

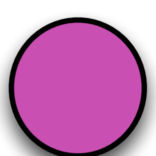
Tolerating Byzantine failures



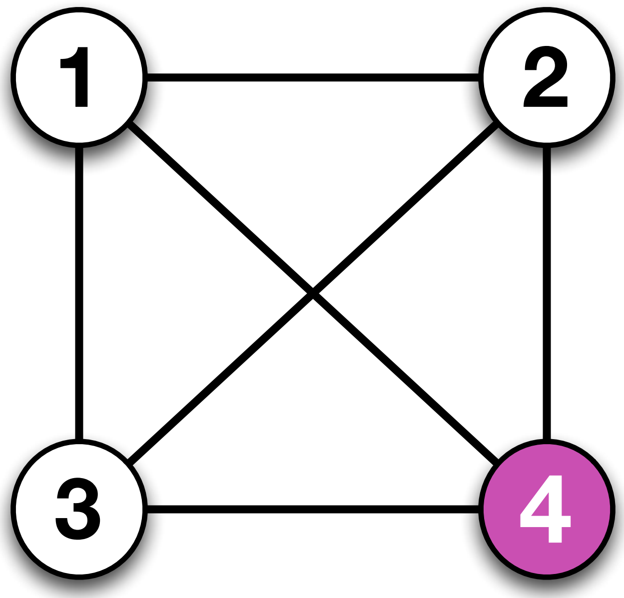
Assume that at most f nodes may be **Byzantine**.

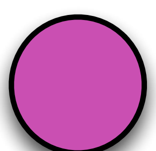
Tolerating Byzantine failures



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

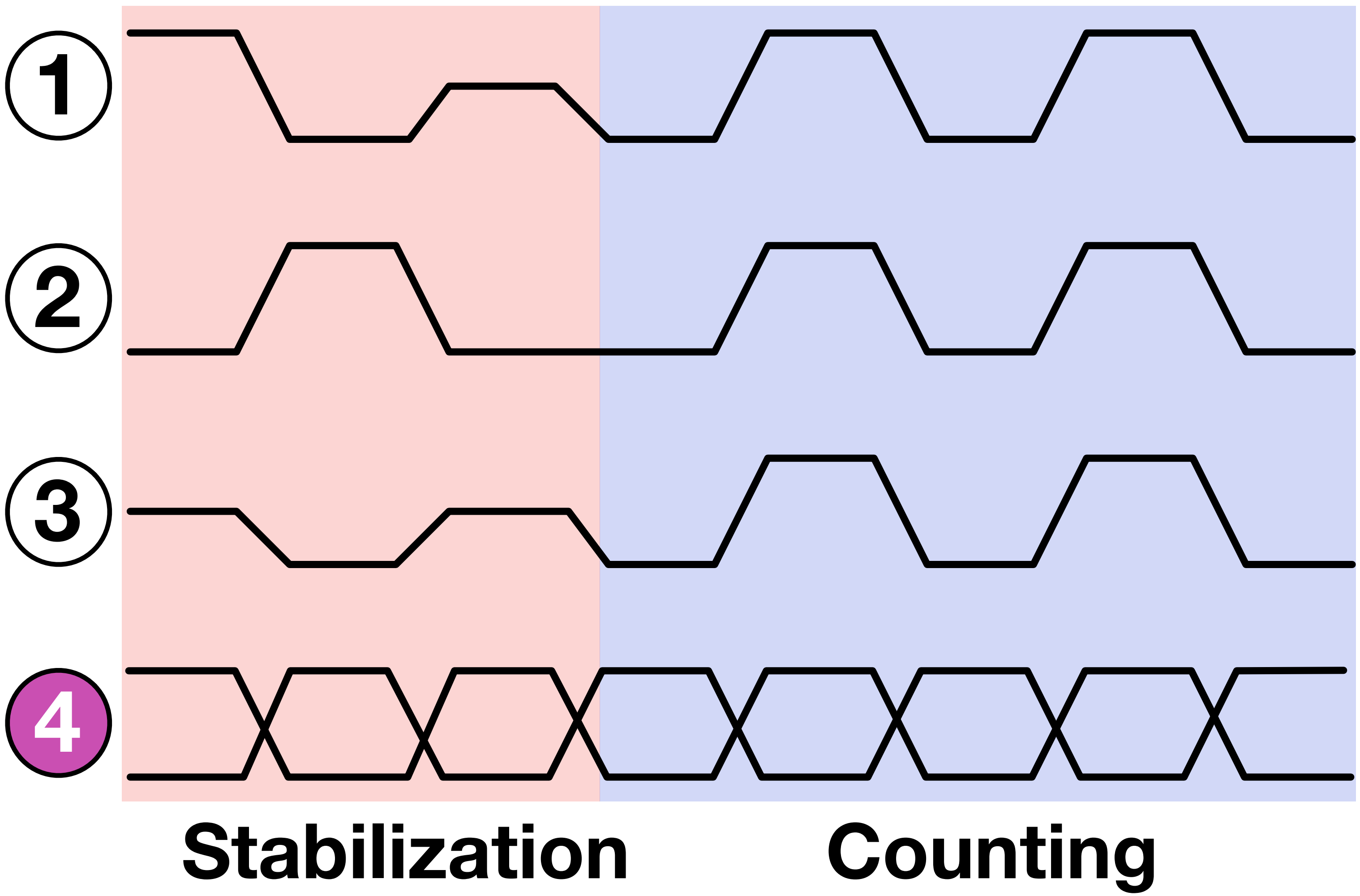
Tolerating Byzantine failures



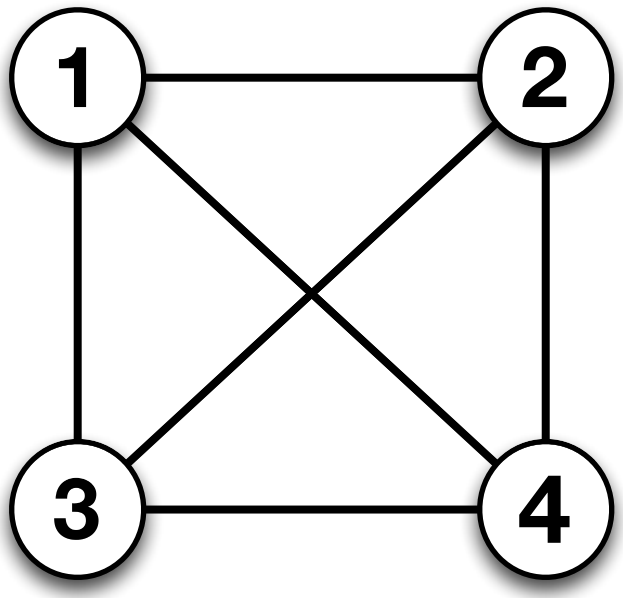
 can send *different* messages to non-faulty nodes!

Note: Easy if self-stabilization is not required!

Fault-tolerant counting



The model with failures



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

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?
  - $t > f$

Pease et al., 1980

Fischer & Lynch, 1982

Solving synchronous counting

Deterministic solutions with large *s* known for similar problems (e.g. D. Dolev & Hoch, 2007)

Randomized solutions for counting with small *s* and large *t* in expectation (e.g. Shlomi Dolev's book)

Our work:

Are there *deterministic* algorithms with small *s* and *t*?

Focus on the first non-trivial case $f = 1$

Generalizing from a base case

For any fixed *s*, *f* and *t*:

There is an algorithm **A** for *n* nodes

$$\Downarrow$$

There is an algorithm **B** for *n*+1 nodes with same *s*, *f* and *t*

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	$3^{324} \approx 10^{154}$

We need a clever way to do the search!

The high-level idea

- Express the existence of an algorithm as a finite combinatorial problem
- Solve a base case that implies a general solution
- **SAT solvers** solve the decision problem

SAT solving

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

Example 1: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$

SAT solving

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

Example 1: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$

Satisfiable!

$$x_1 = 0$$
$$x_2 = 0$$
$$x_3 = 1$$

SAT solving

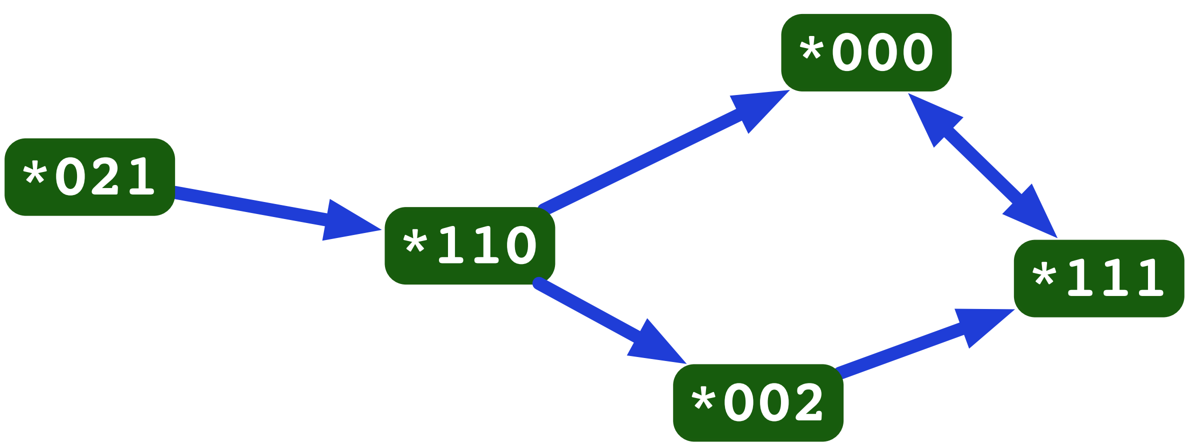
- NP-hard
- Surprisingly fast in practice
- Complete: proves **YES** and **NO** instances
- Several solvers available

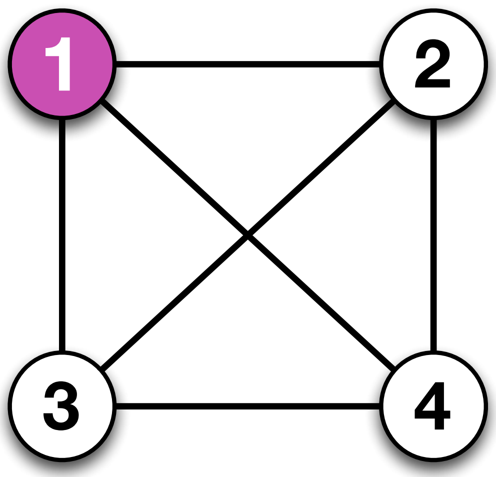
Verification is easy

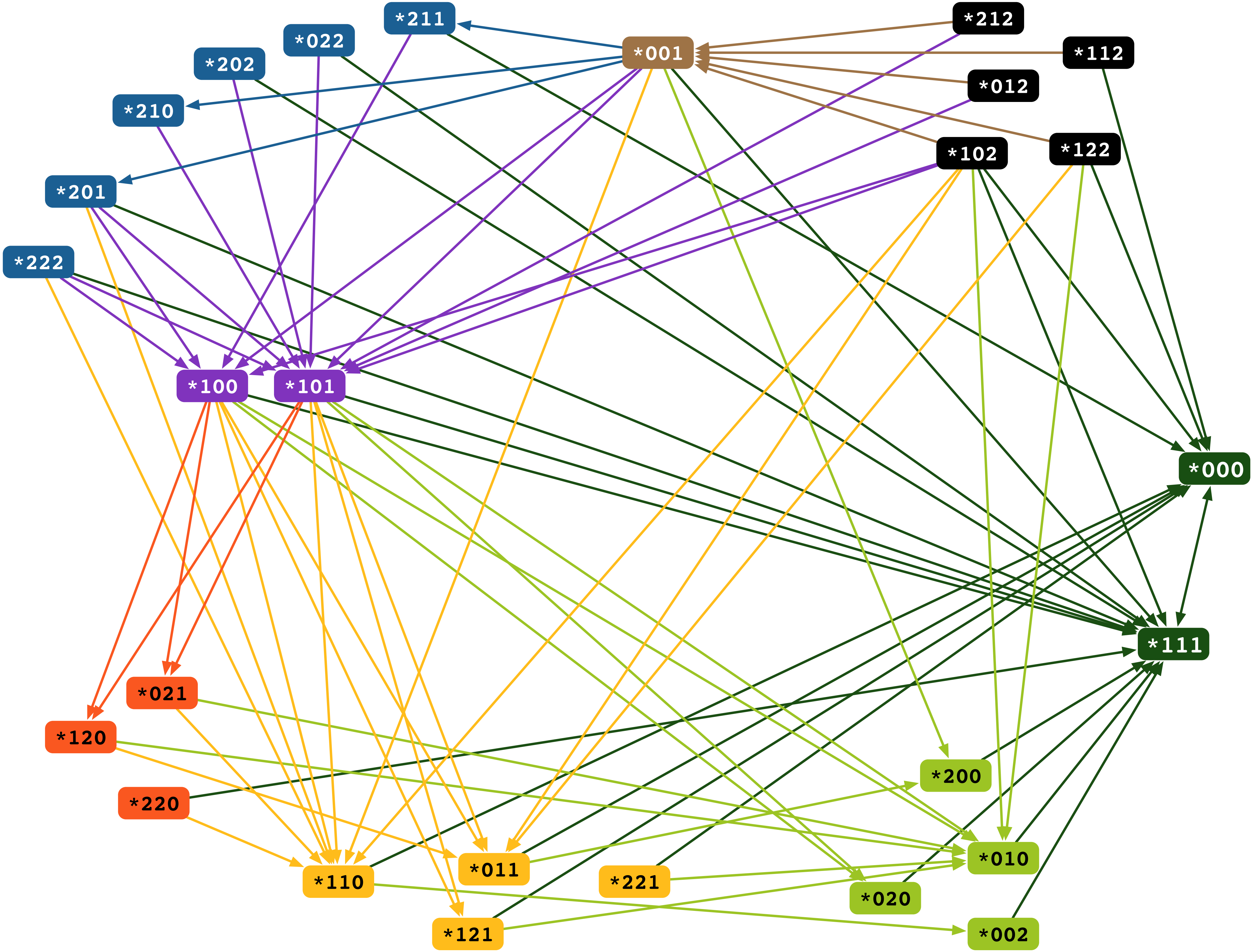
- Let F be a set of faulty nodes, $|F| \leq f$
- Construct a *state graph* G_F from **A**:

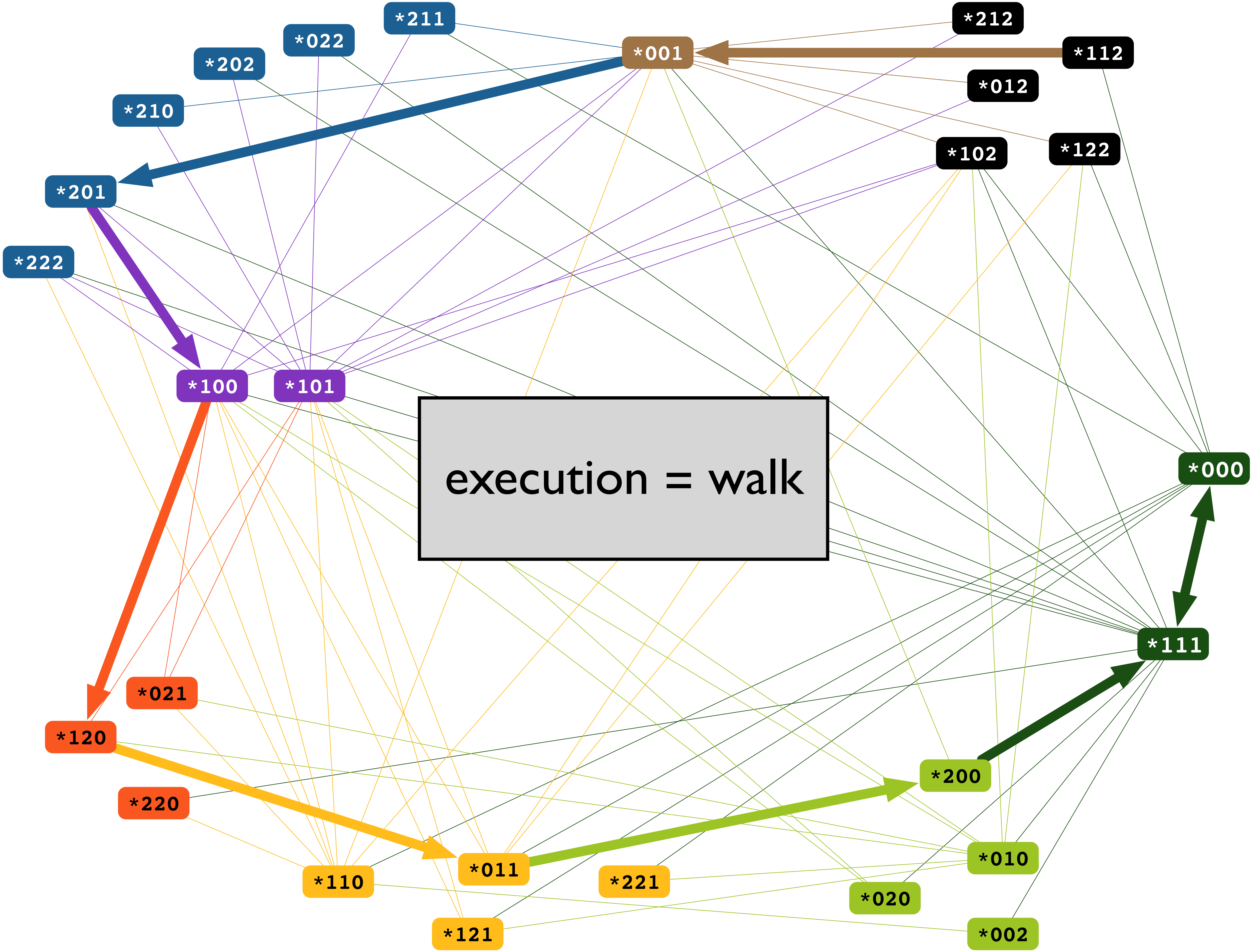
  Nodes = actual states

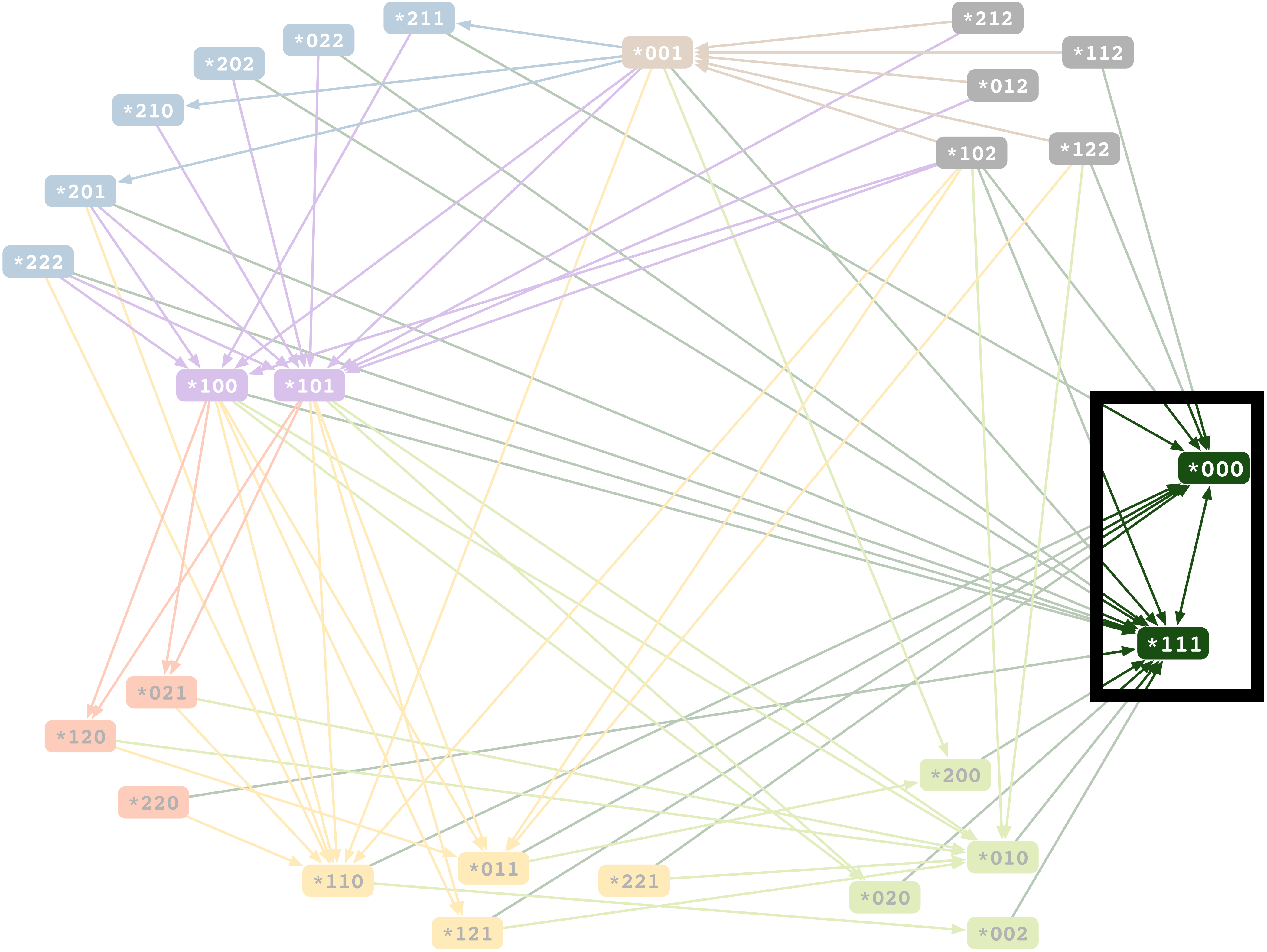
  Edges = possible state transitions



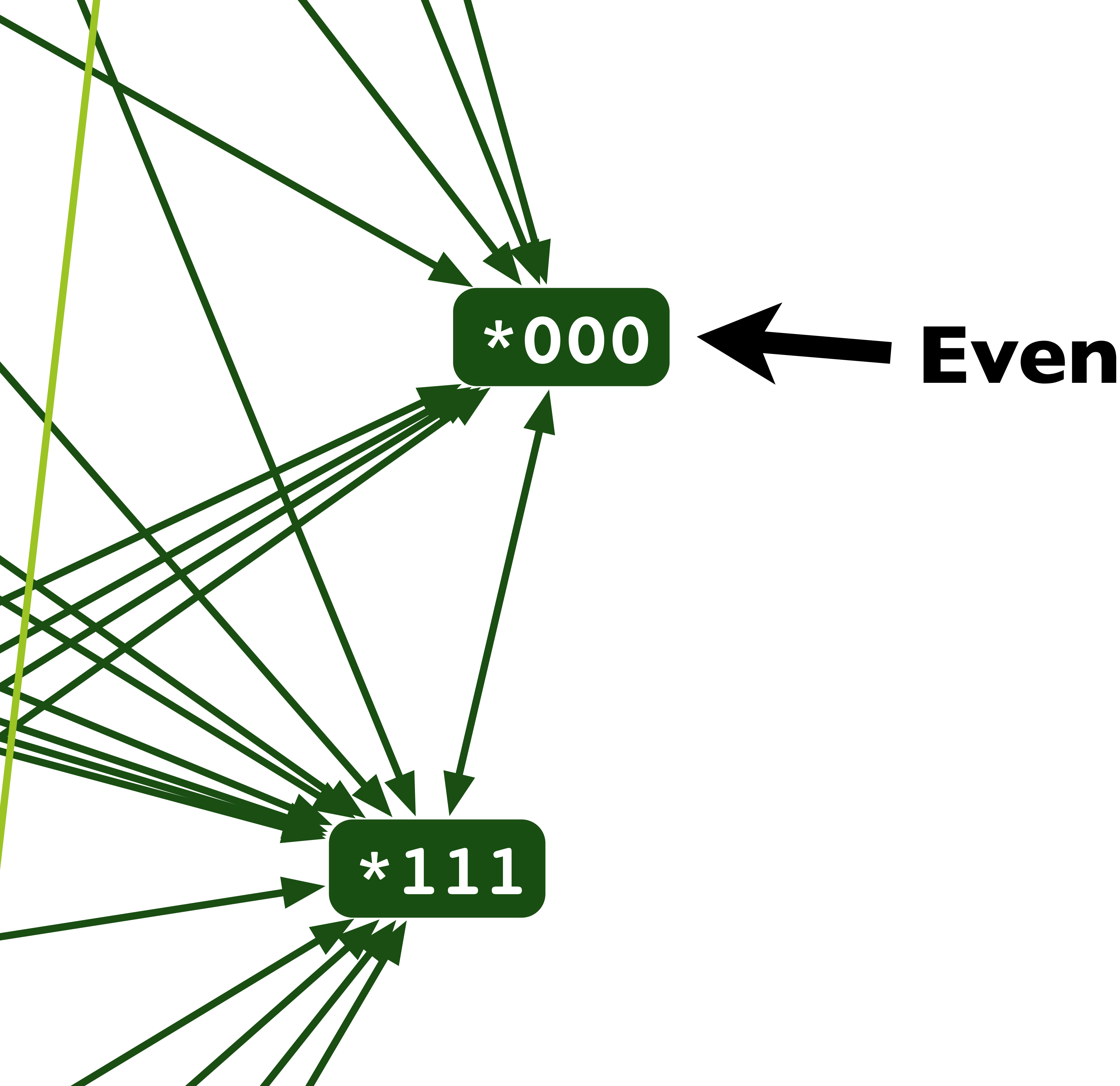


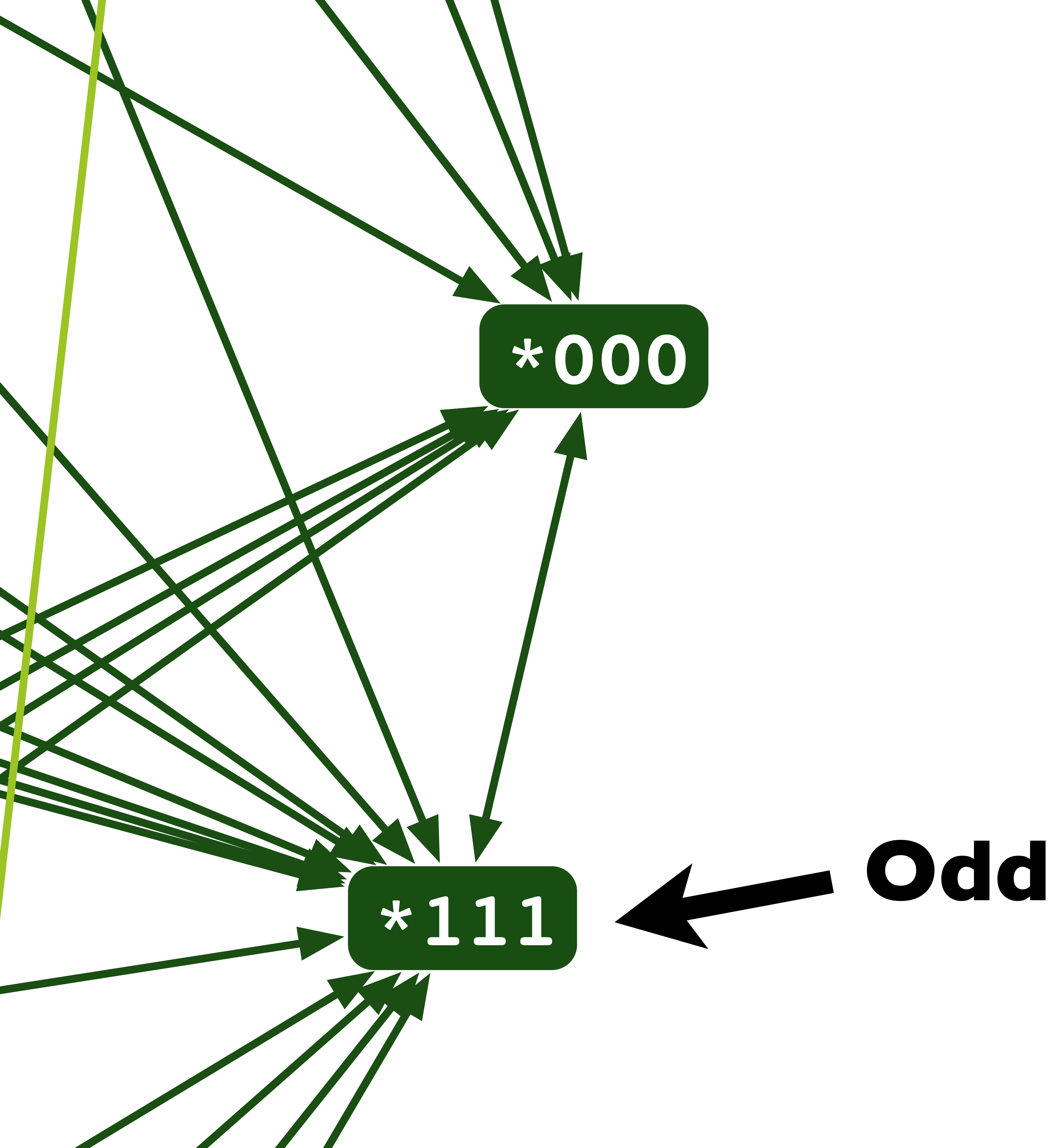

*211
*022
*202
*210
*201
*222
*001
*212
*112
*012
*102
*122
*100
*101
*000
*111
*021
*120
*220
*200
*010
*110
*011
*221
*020
*121
*002

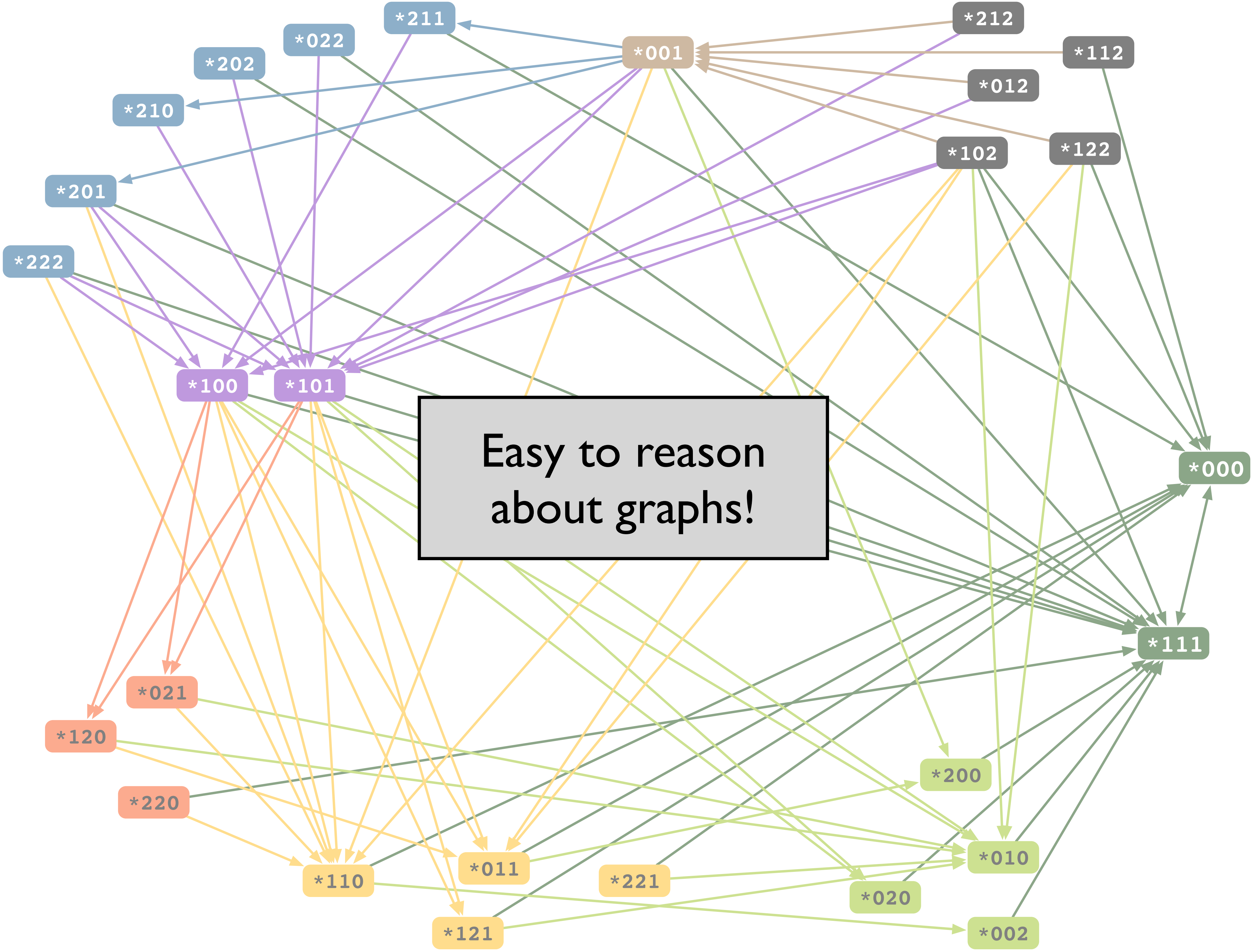

execution = walk
*211
*022
*202
*210
*201
*222
*100
*101
*001
*212
*112
*012
*102
*122
*000
*111
*021
*120
*220
*110
*011
*121
*221
*200
*010
*020
*002

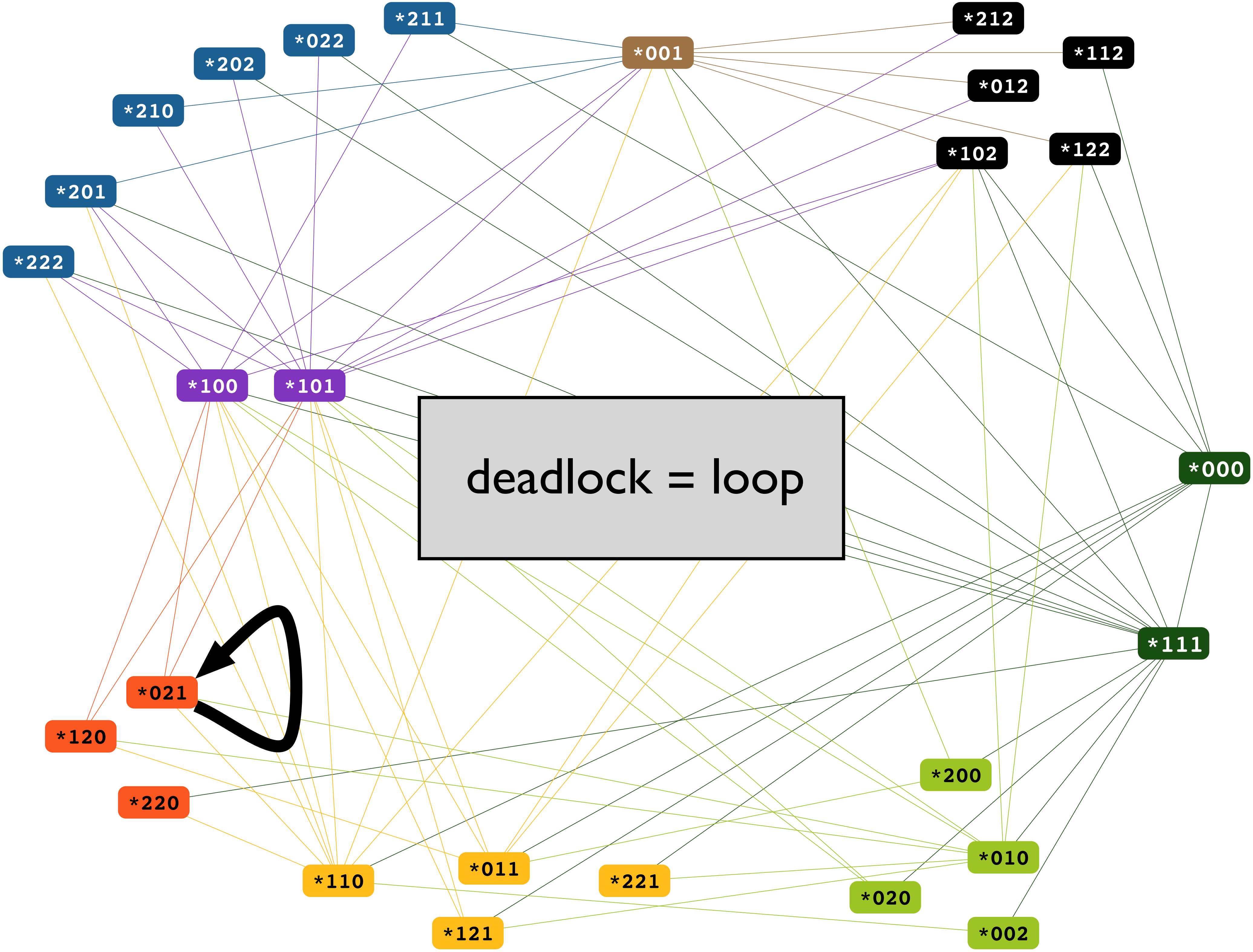


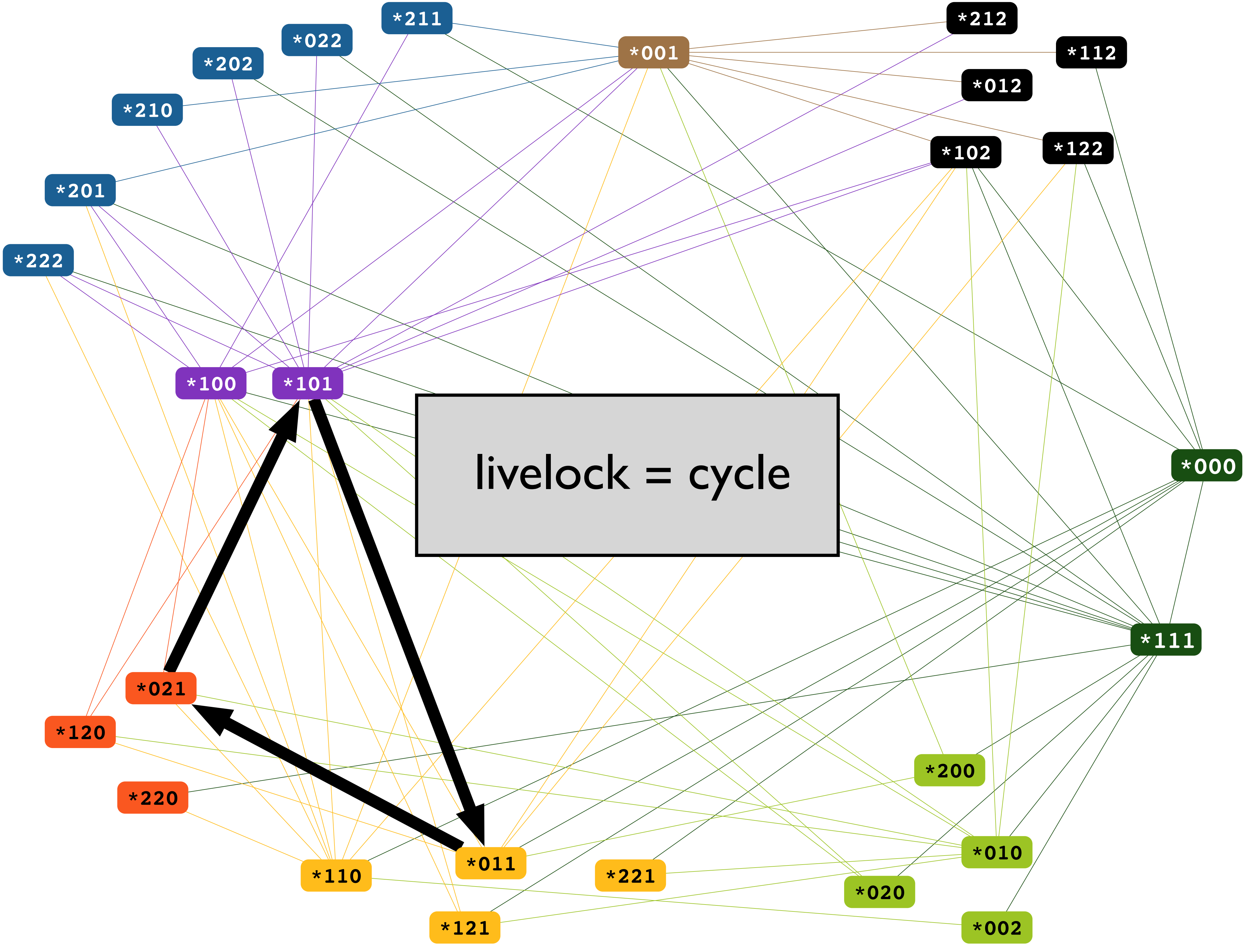
*211
*022
*202
*210
*201
*222
*001
*212
*112
*012
*102
*122
*100
*101
*000
*111
*021
*120
*220
*200
*010
*011
*110
*221
*020
*121
*002


*000
Even
*111


*000
*111
Odd


*211
*212
*022
*001
*112
*202
*012
*210
*102
*122
*201
*222
*100
*101
Easy to reason about graphs!
*000
*111
*021
*120
*200
*220
*010
*011
*110
*221
*020
*121
*002


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


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

Verification is easy

A is **correct**	$\Leftrightarrow$	Every G_F is **good**
no deadlocks	$\Leftrightarrow$	G_F is loopless
stabilization	$\Leftrightarrow$	All nodes have a path to **0**
counting	$\Leftrightarrow$	$\{\mathbf{0}, \mathbf{1}\}$ is the only cycle

From verification to synthesis

The encoding uses the following variables:

$$x_{i,u,s} \quad \Leftrightarrow \quad A_i(u) = s$$

$$e_{q,r} \quad \Leftrightarrow \quad \text{edge } (q,r) \text{ exists}$$

$$p_{q,r} \quad \Leftrightarrow \quad \text{path } q \rightsquigarrow r \text{ exists}$$

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

Main results, $f = 1$

If $4 \leq n \leq 5$:

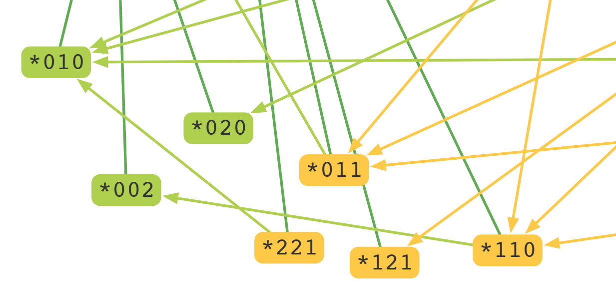
- **lower bound:** no 2-state algorithm
- **upper bound:** 3 states suffice

If $n \geq 6$:

- 2 states always suffice

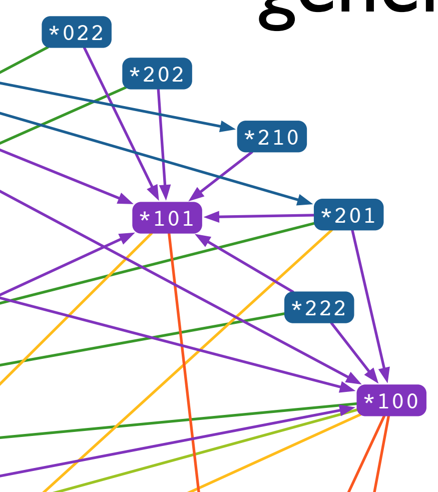
Summary

- We have algorithms that use the optimal number of states for any n and $f = 1$
- Computational techniques useful in design of fault-tolerant algorithms
- Solve a base case using computers; let people generalize



Summary

- We have algorithms that use the optimal number of states for any *n* and $f = 1$
- Computational techniques useful in design of fault-tolerant algorithms
- Solve a base case using computers; let people generalize



Thanks!