# Exploiting Belief State Structure in Graph Search

Jason Wolfe & Stuart Russell
University of California - Berkeley

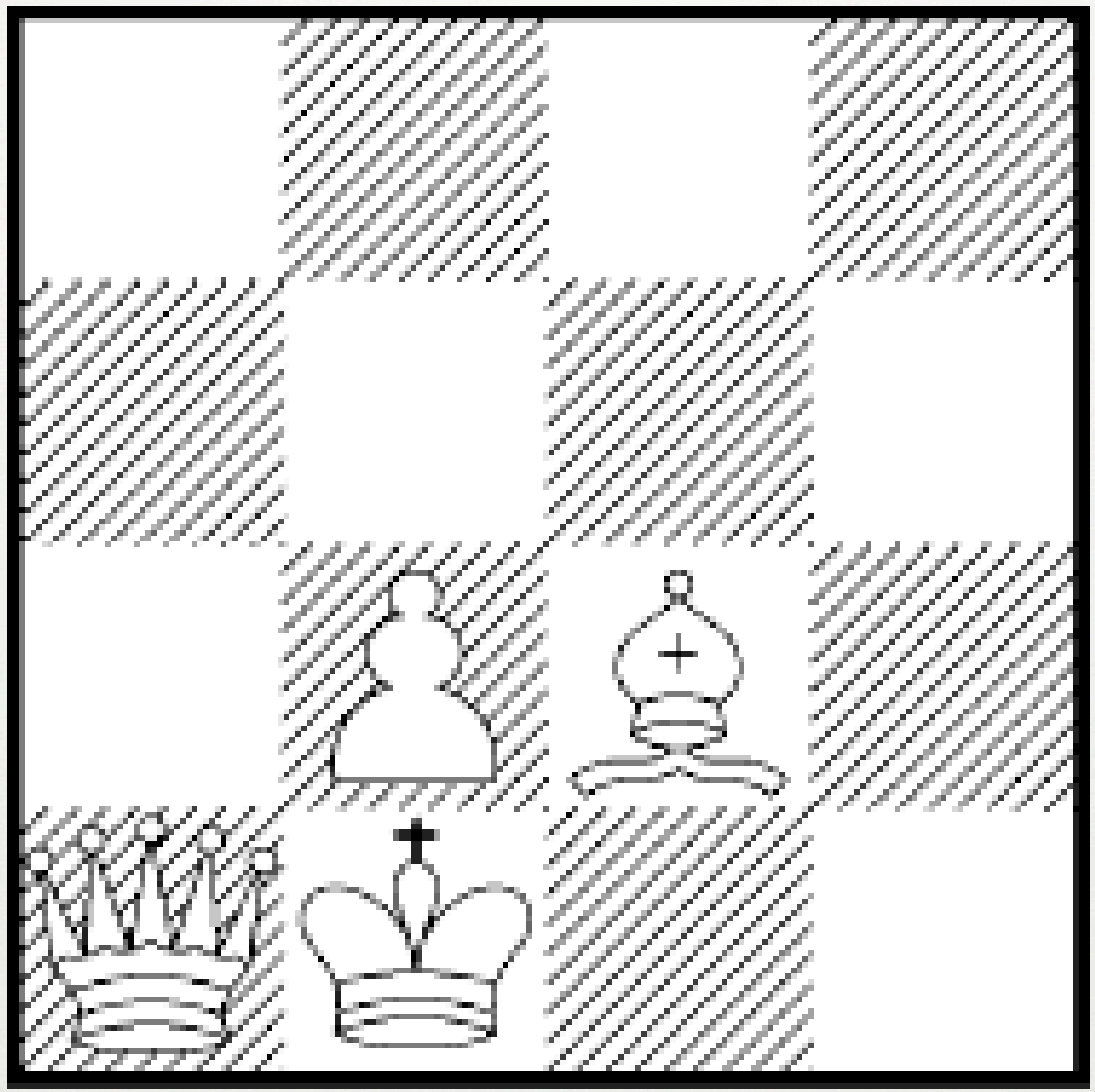
# Introduction

- Recognizing already-solved subproblems can be essential for efficient search, e.g.,
  - A* graph search, α-β with transposition table
  - Subsumption in theorem proving
  - "Nogoods" in CSP & SAT solving, planning
- We present a novel, effective application to partially observable games & planning
- Generalize notion of graph search to incorporate subsumption relationships

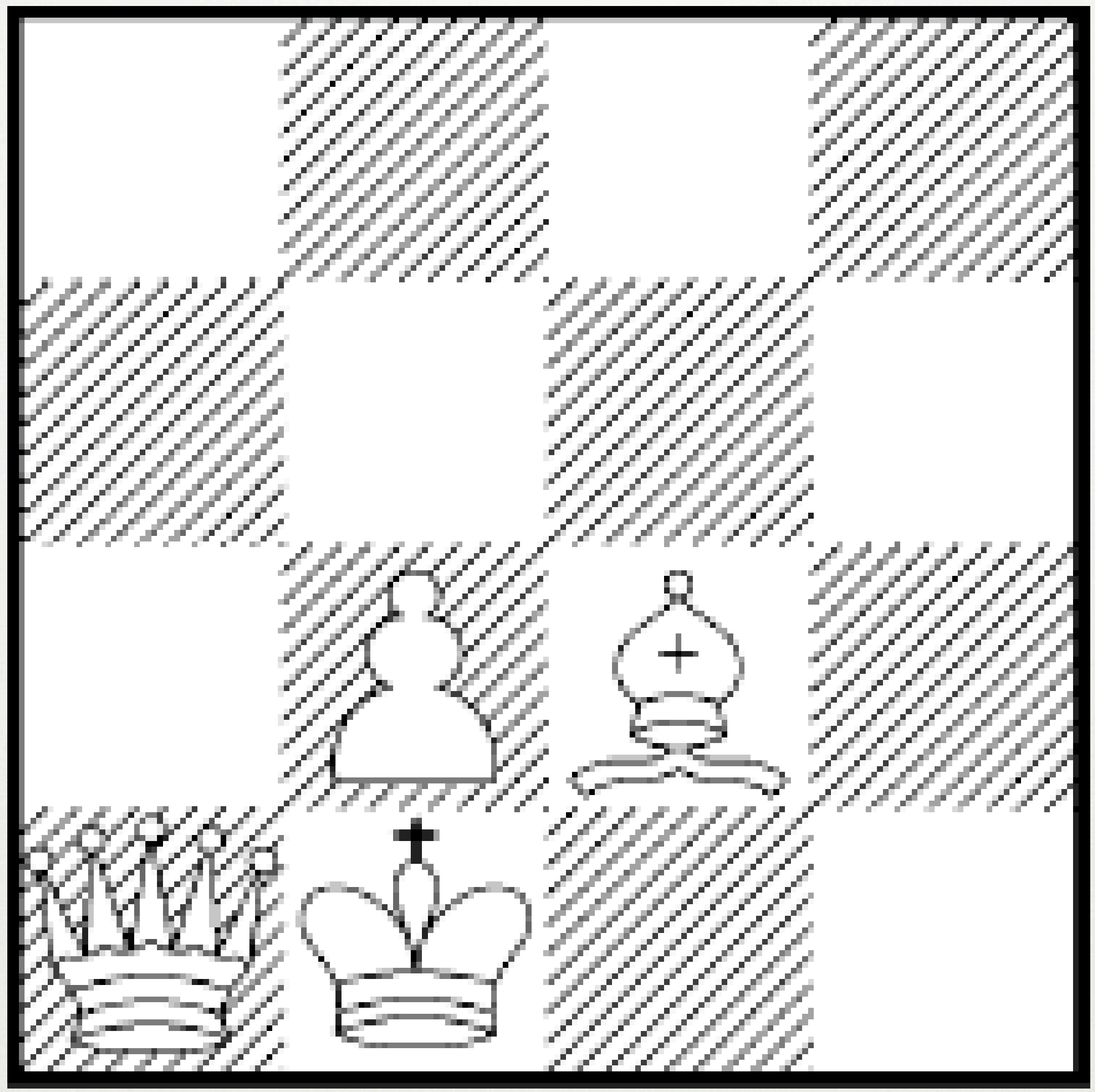
# Motivation

- Working on finding Kriegspiel checkmates
  - chess -- but opponent pieces and moves secret
  - symmetric percepts: illegal / check / capture / ...
  - players attempt moves until one found legal ("try-until-feasible" property)
- Branching factor is huge (worst case ~30!), but we want to scale to 7-ply and beyond
- Developed methods here to help tame this branching factor

# (4x4) Kriegspiel

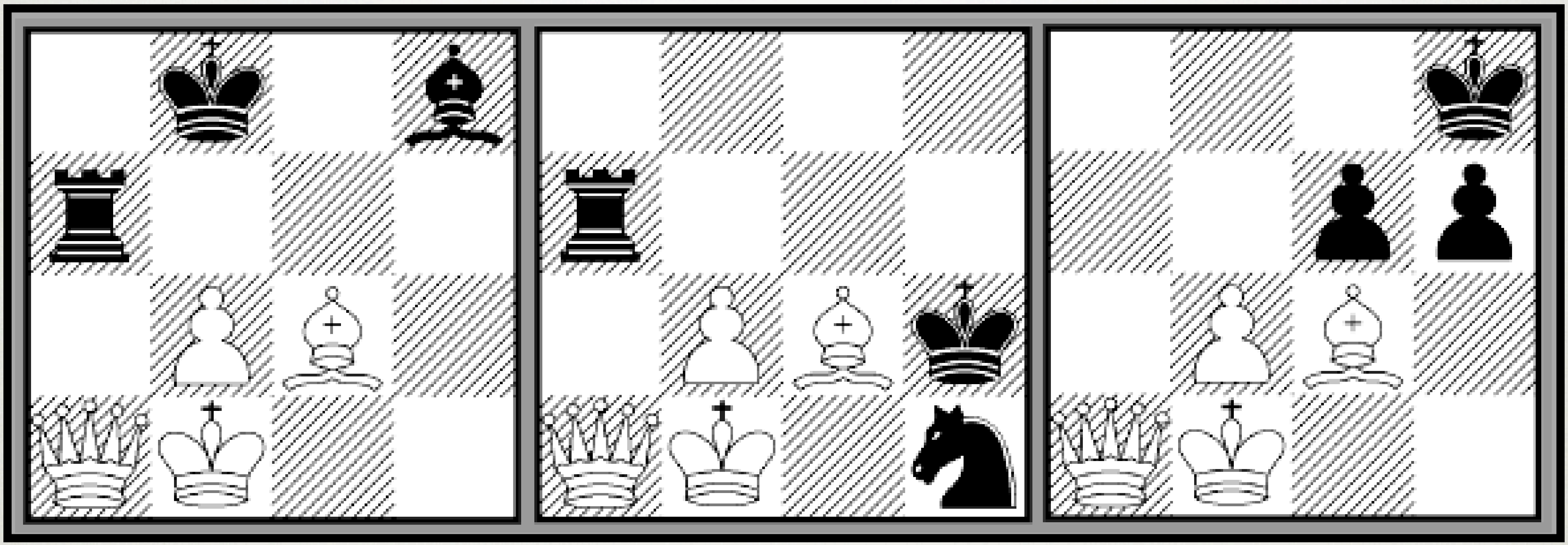


# (4x4) Kriegspiel



White to move and checkmate within 3 turns

# A (4x4) Kriegspiel checkmate



White to move and checkmate within 3 turns

# Outline

- Background:
  - Partially observable planning & games
  - Belief-state AND/OR trees
  - Search algorithms **DFS** and **DBU**
- Exploiting related belief states:
  - Graph versions of **DFS** and **DBU**
  - Data structures for subset lookups
- Experimental results & conclusions

# Partially Observable Planning

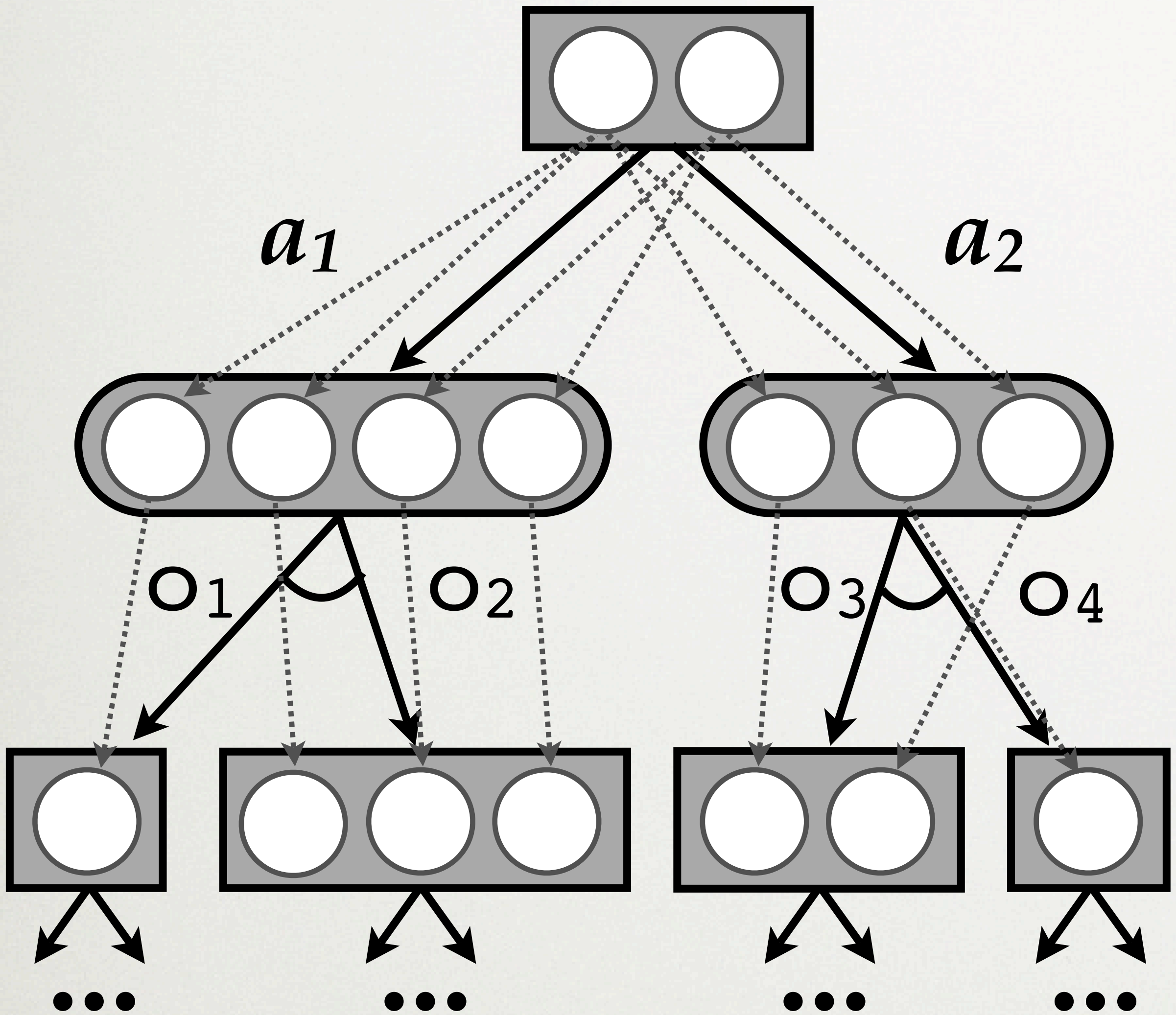
- World state is partially observable (PO)
- Actions may be nondeterministic
- Plans may be contingent on observations
- Goal: strong, fixed-depth (acyclic) plans
  - Guaranteed to reach goal in fixed # actions
  - Find by searching AND-OR tree where nodes correspond to agent's belief states

[cf. Bertoli *et. al.* (2001), Sakuta and Iida (2001)]

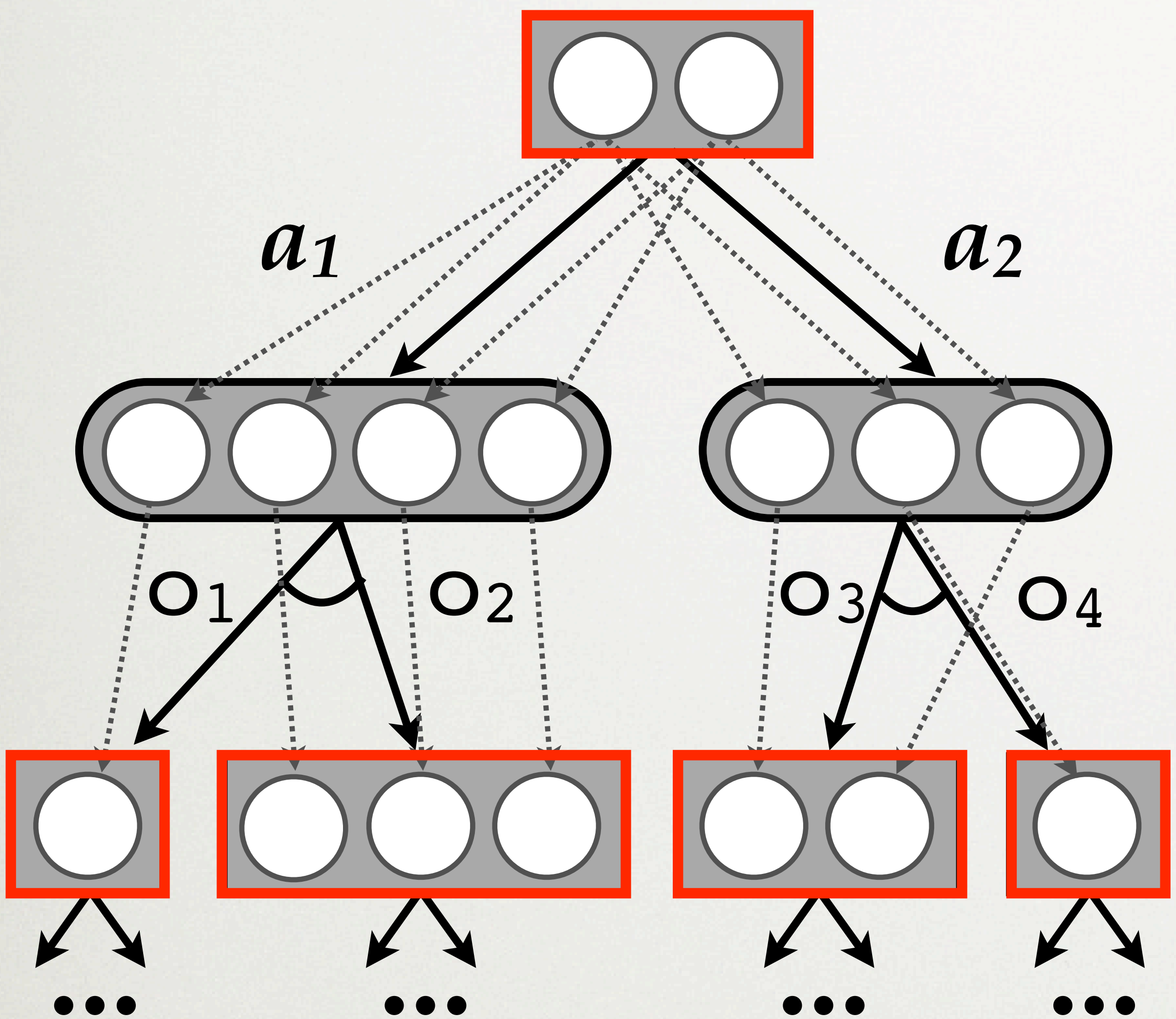
# Guaranteed Wins in PO Games

- Strategies that guarantee the optimal payoff within *k* moves.
- Isomorphic to strong acyclic plans
  - Treat other agents like nondeterminism
  - Works because win must be guaranteed (i.e., work for all possible opponent strategies)

# Belief-state and/or Trees

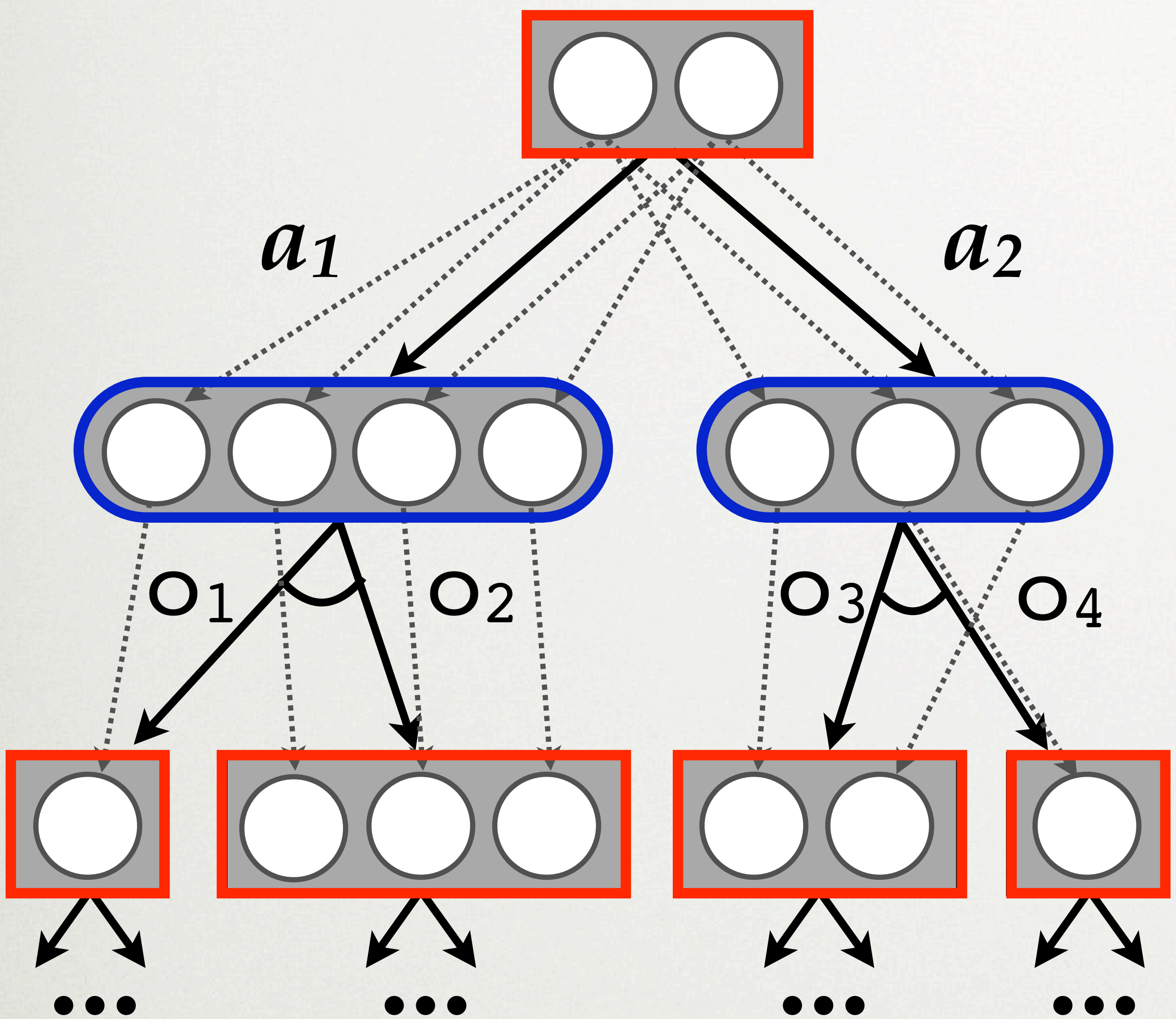


# Belief-state AND/OR Trees




- OR-nodes
  - agent choice points
  - one child per move
  - child has possible action outcomes
  - proven iff all goal states or some child proven

# Belief-state AND/OR Trees




- OR-nodes
  - agent choice points
  - one child per move
  - child has possible action outcomes
  - proven iff all goal states or some child proven
- AND-nodes
  - observation points
  - one child per obs.
  - children partition states
  - proven iff all children proven

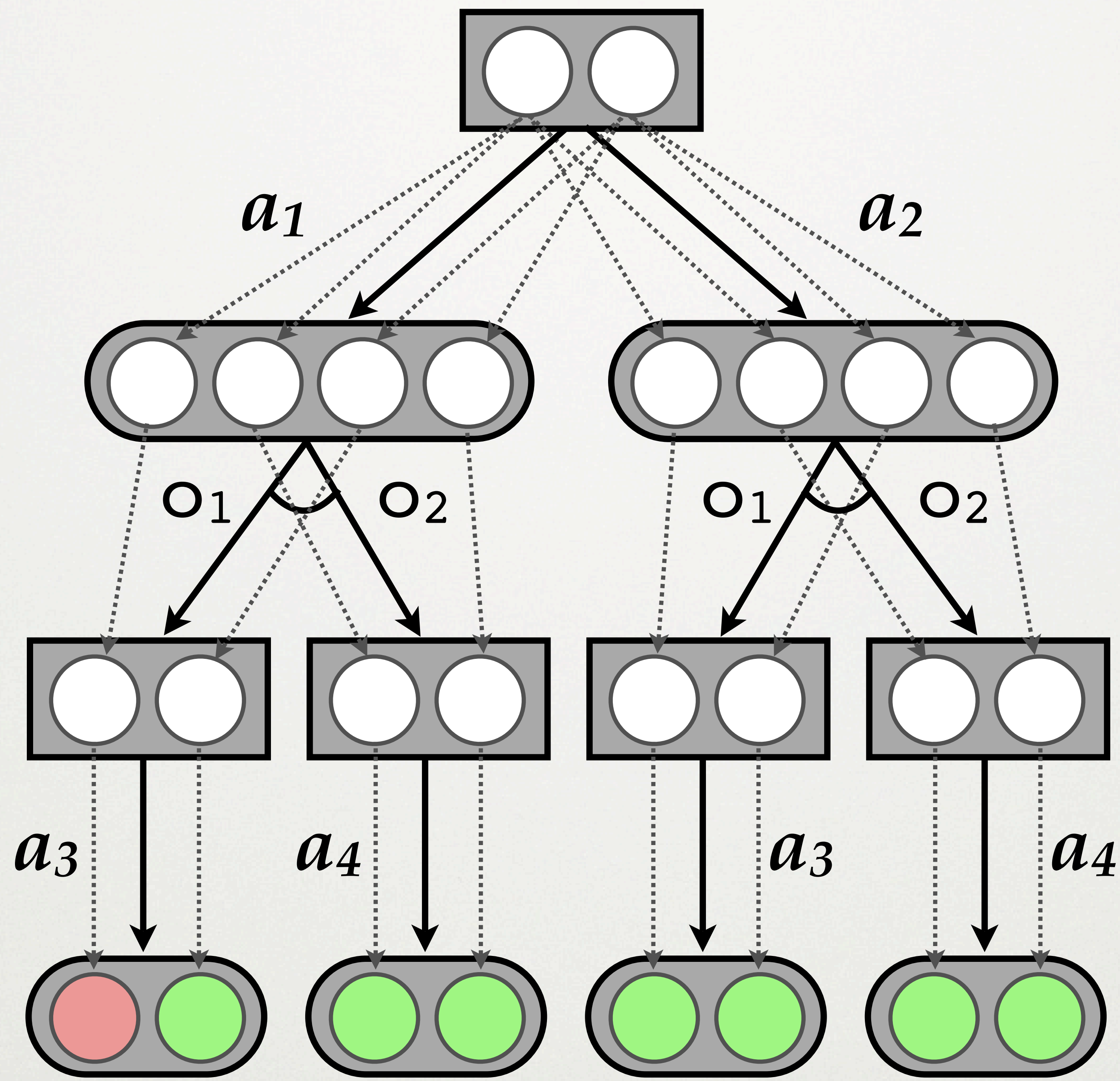
# Search algorithm **DFS**

- Simple approach: execute ordinary AND–OR search algorithm (e.g. **DFS**) on belief-state tree

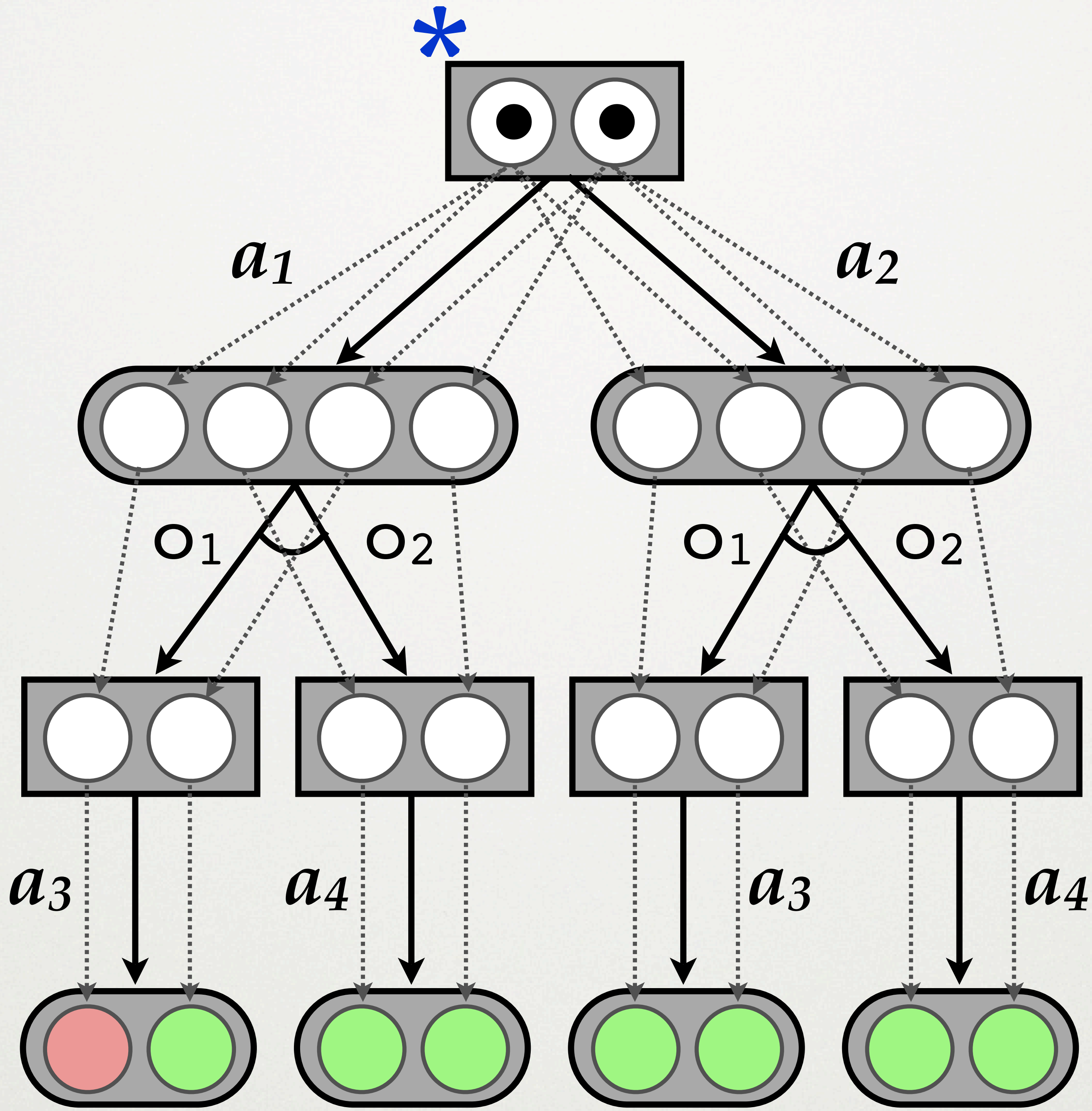
[Sakuta and Iida 2000; Bertoli et. al. 2001; Bolognesi and Ciancarini 2004]

- Essentially just codifies definitions on previous slide (pseudocode provided in paper)

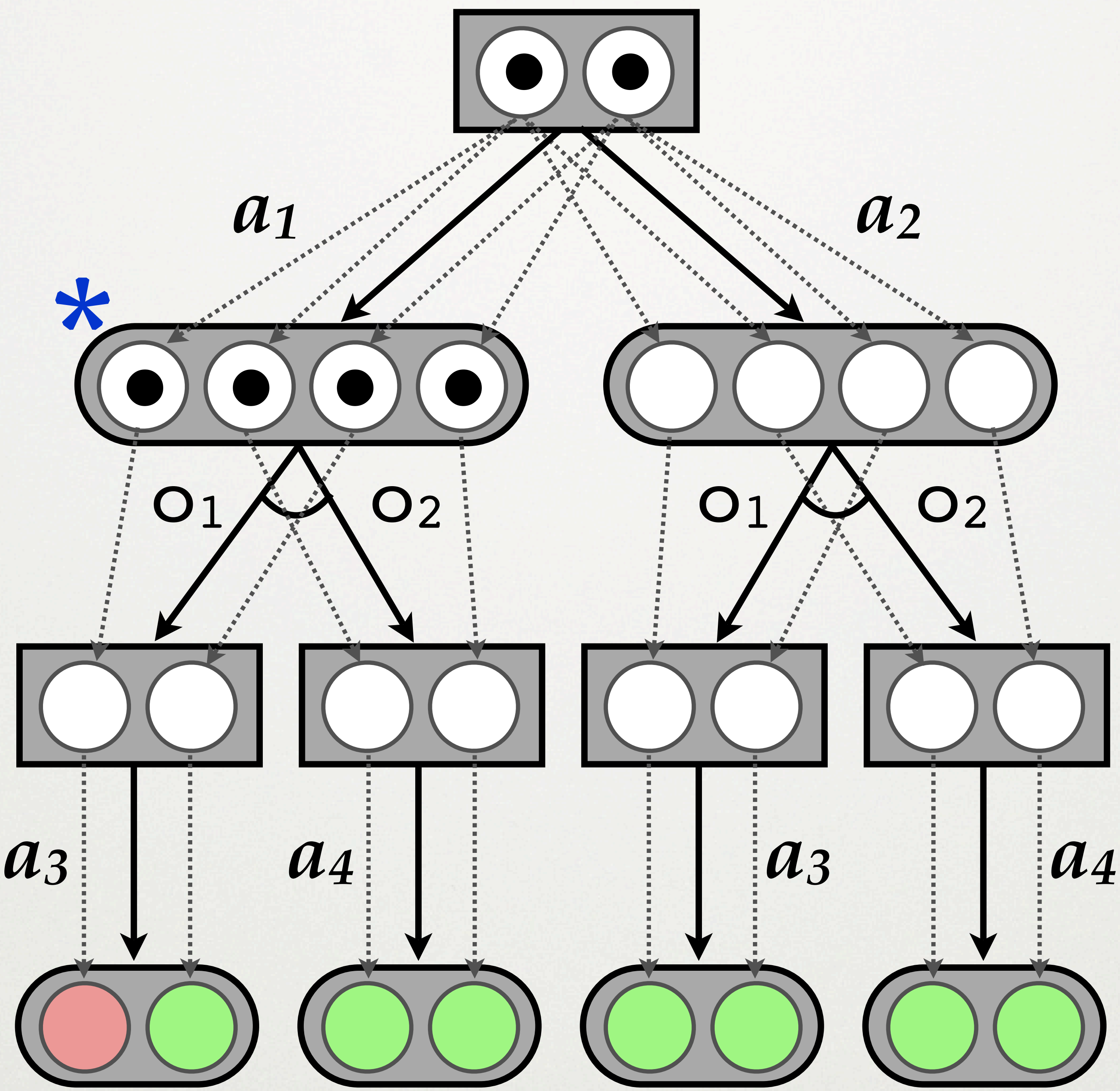
# DFS



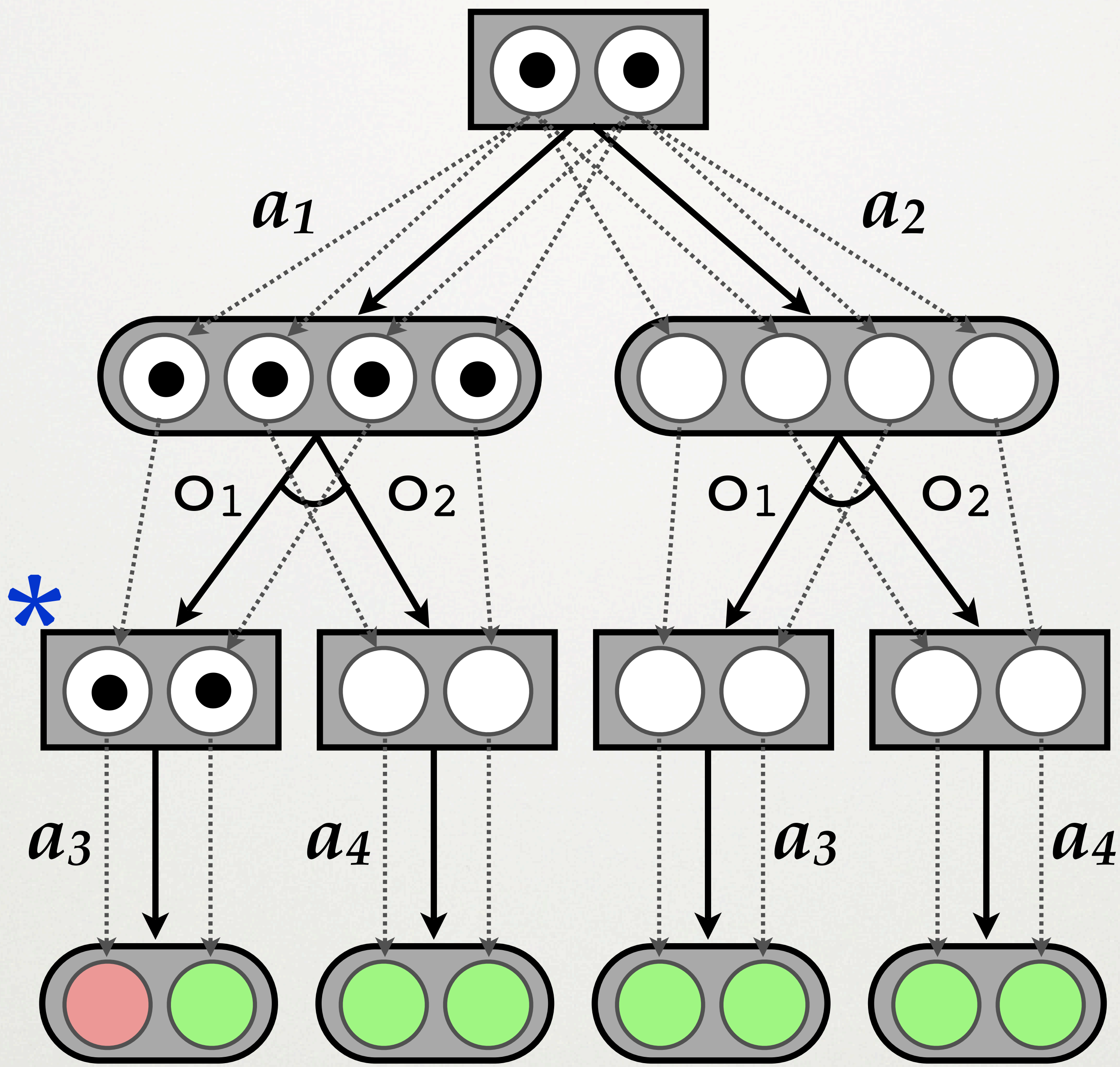
# DFS



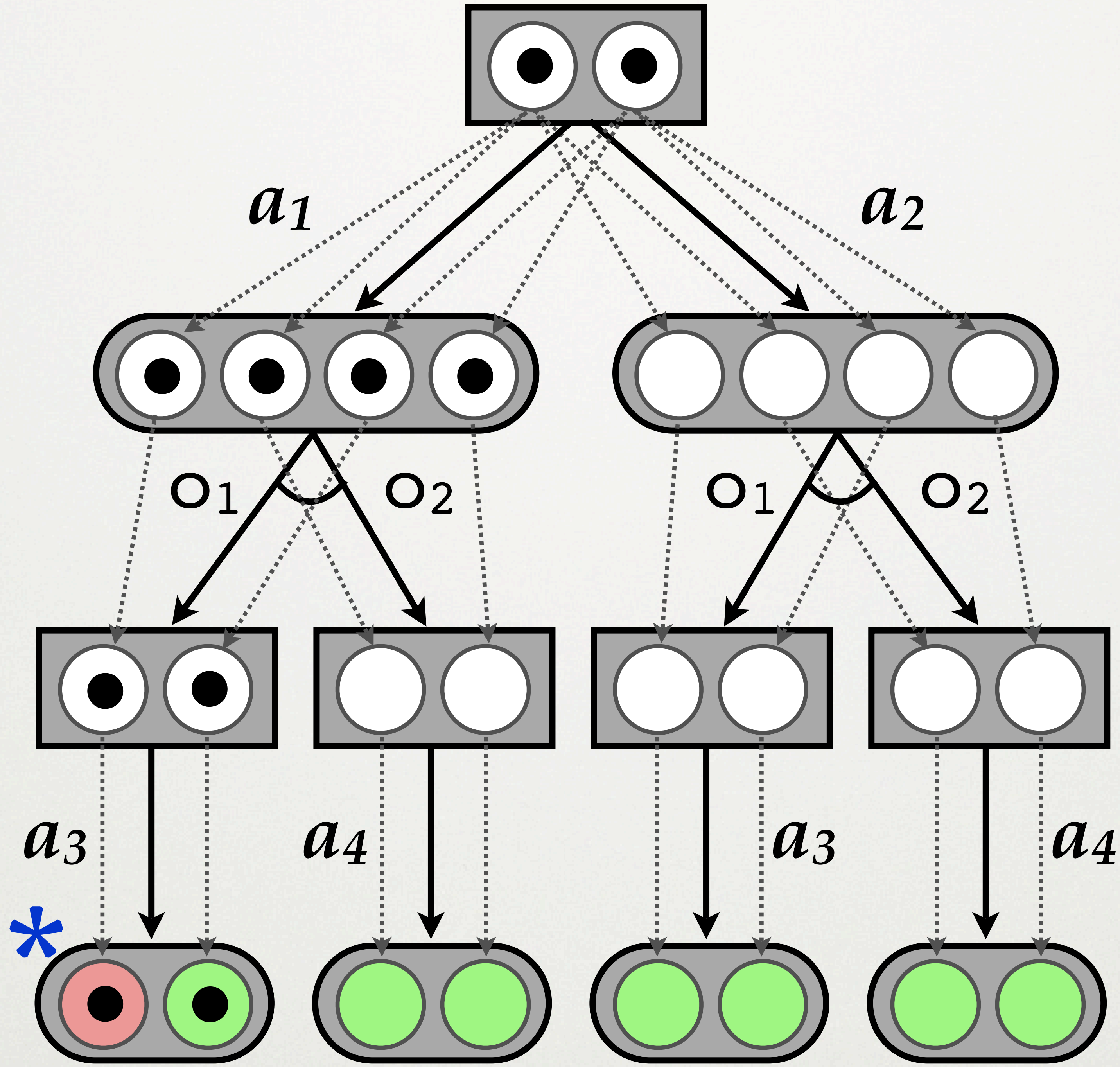
# DFS



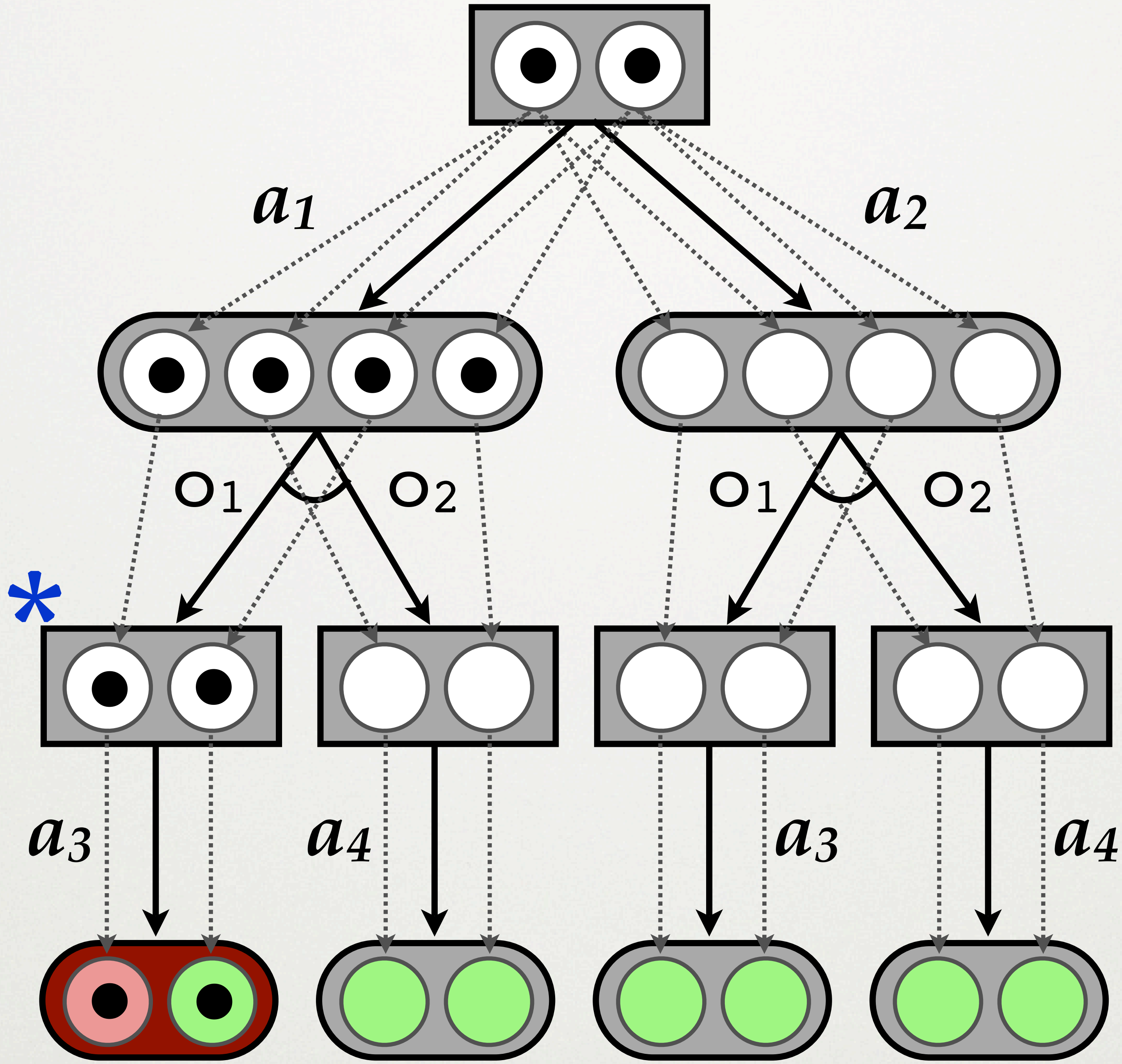
# DFS



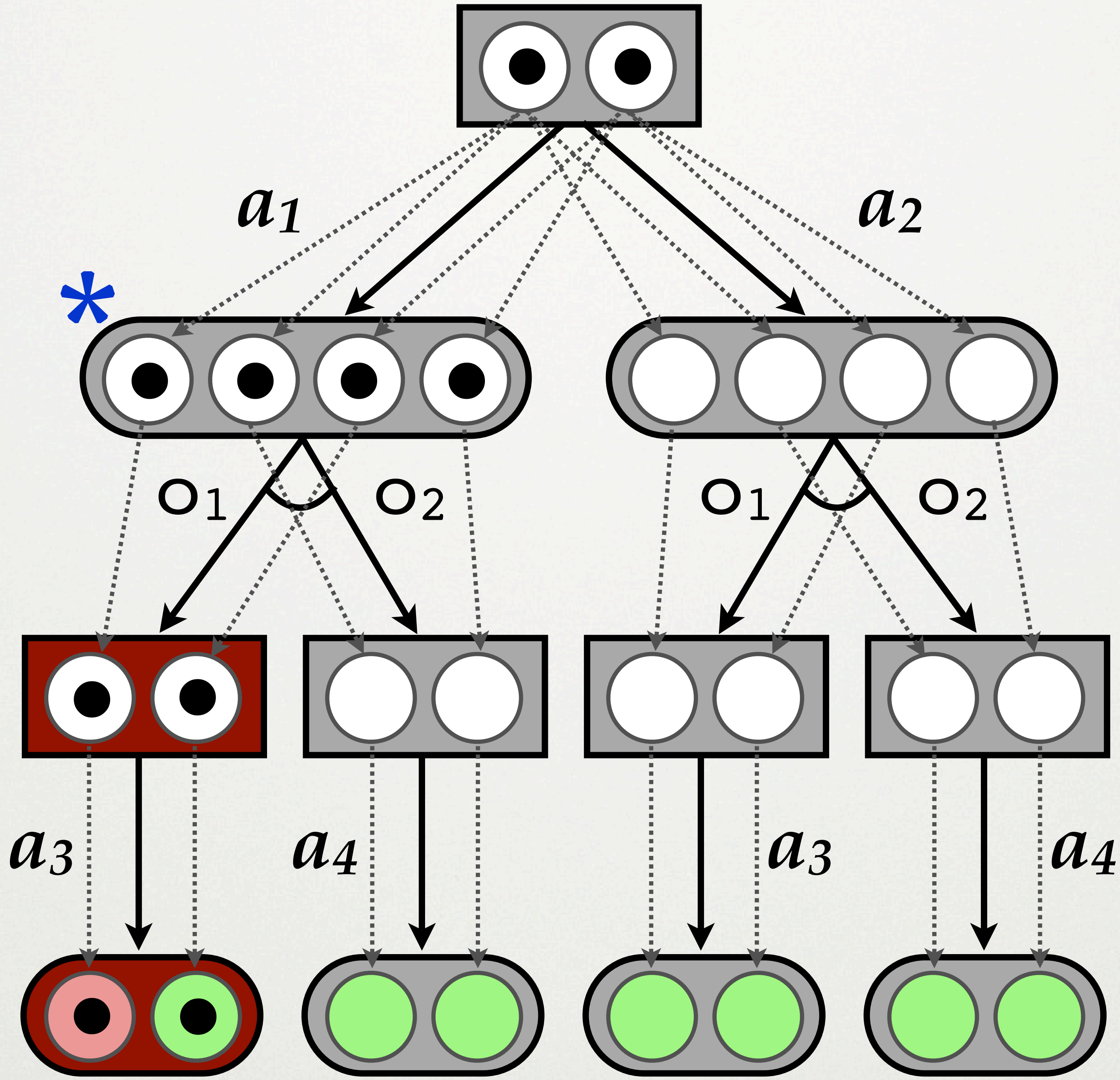
# DFS



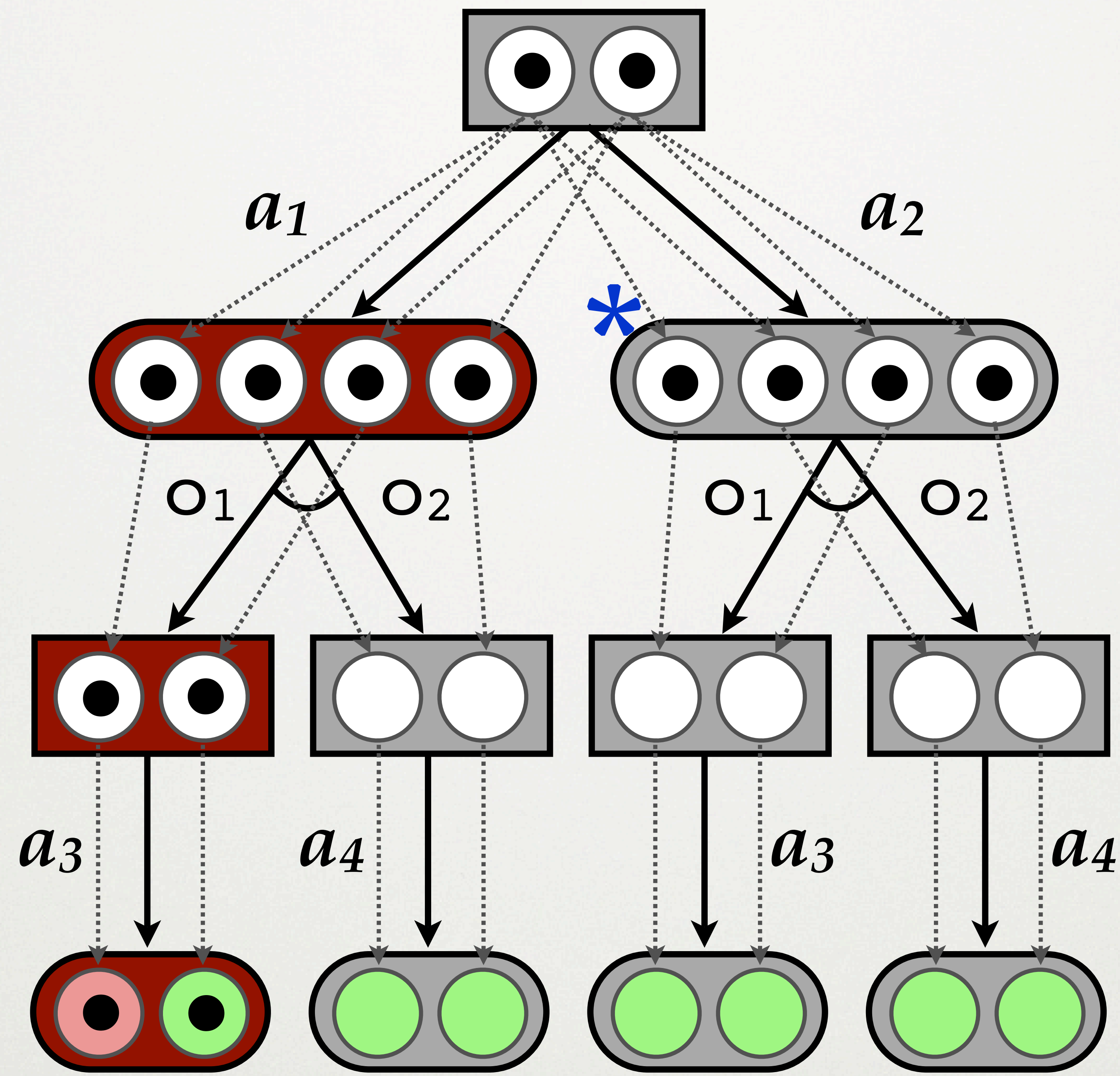
# DFS



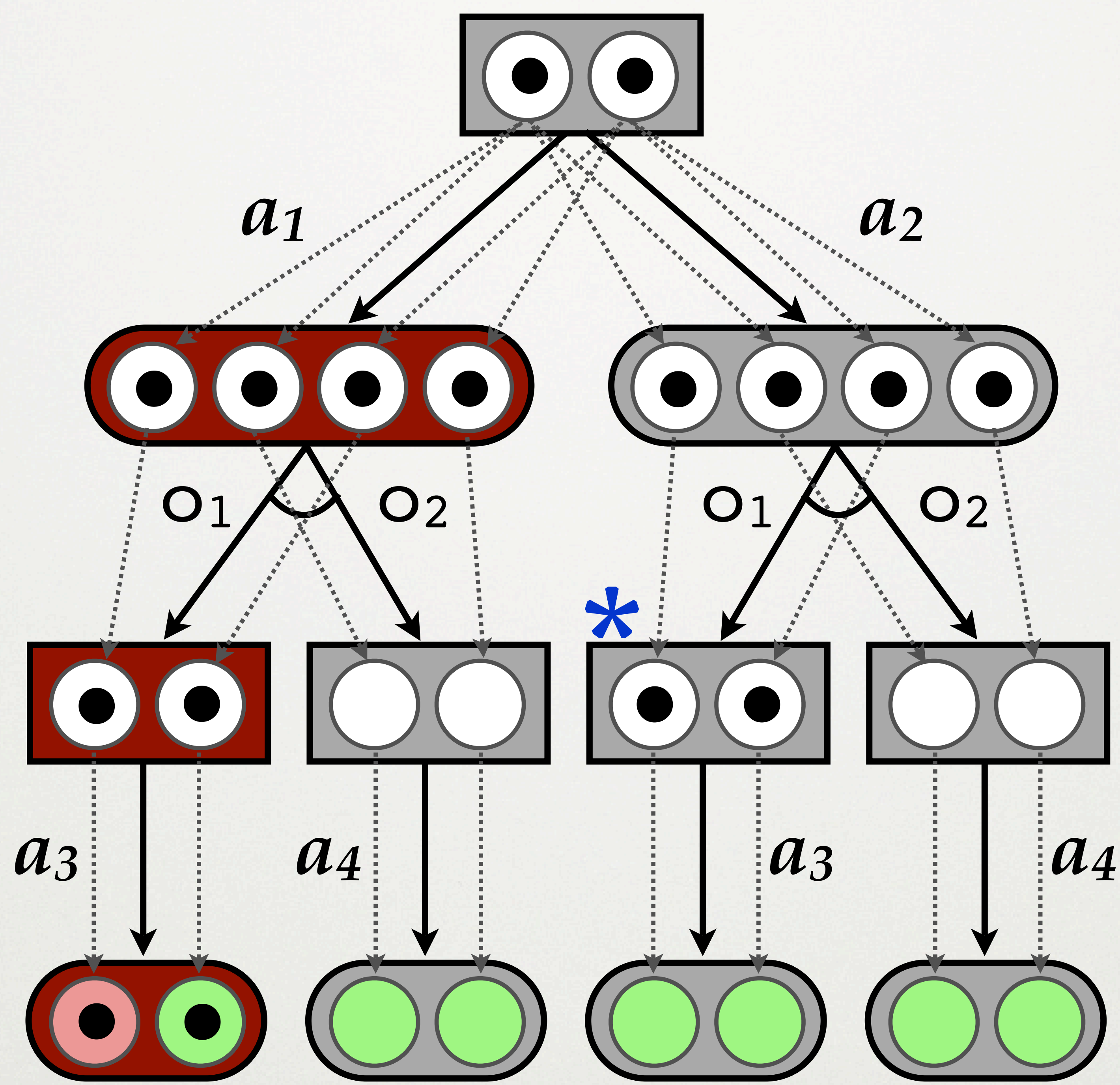
# DFS



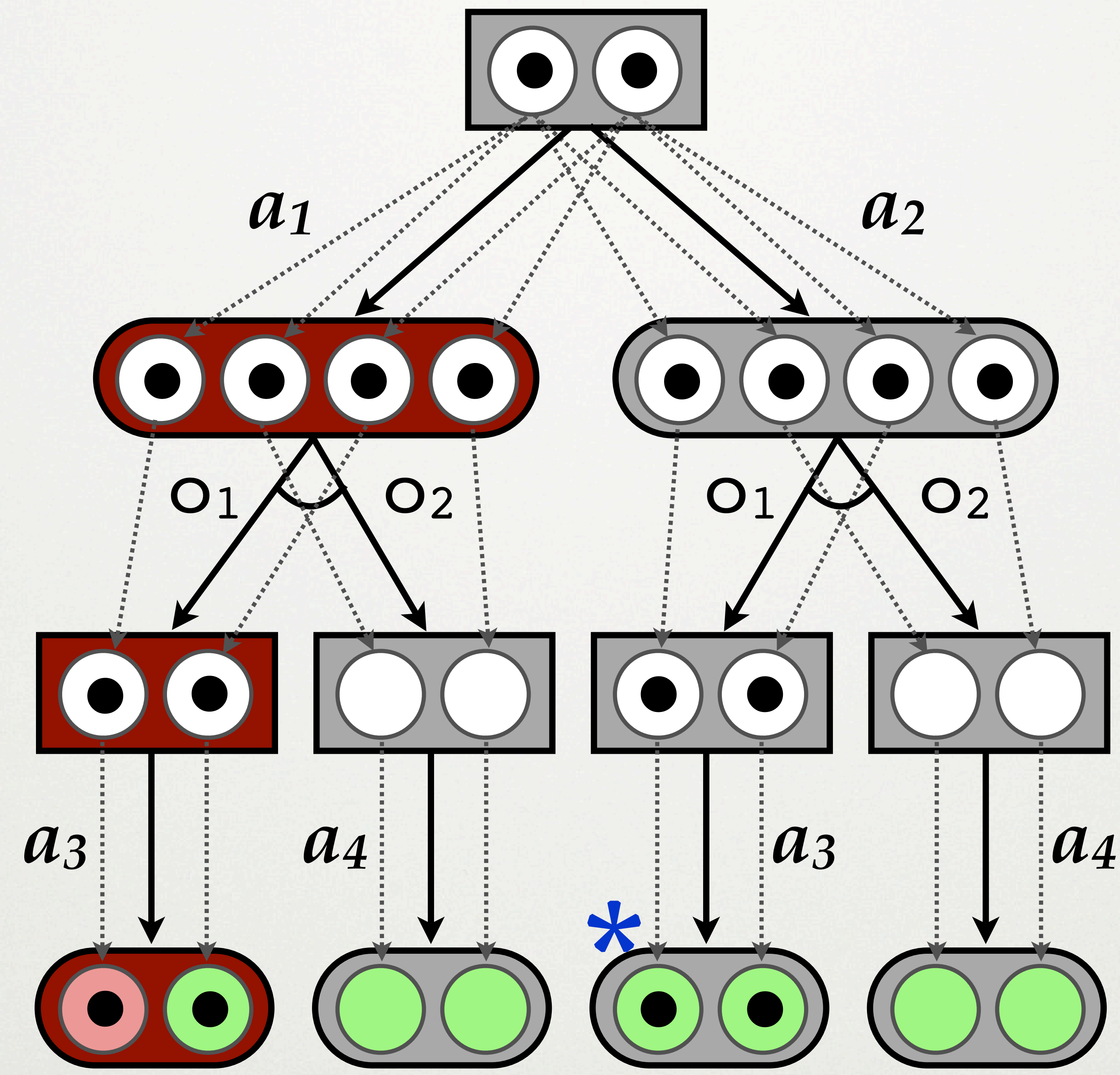
# DFS



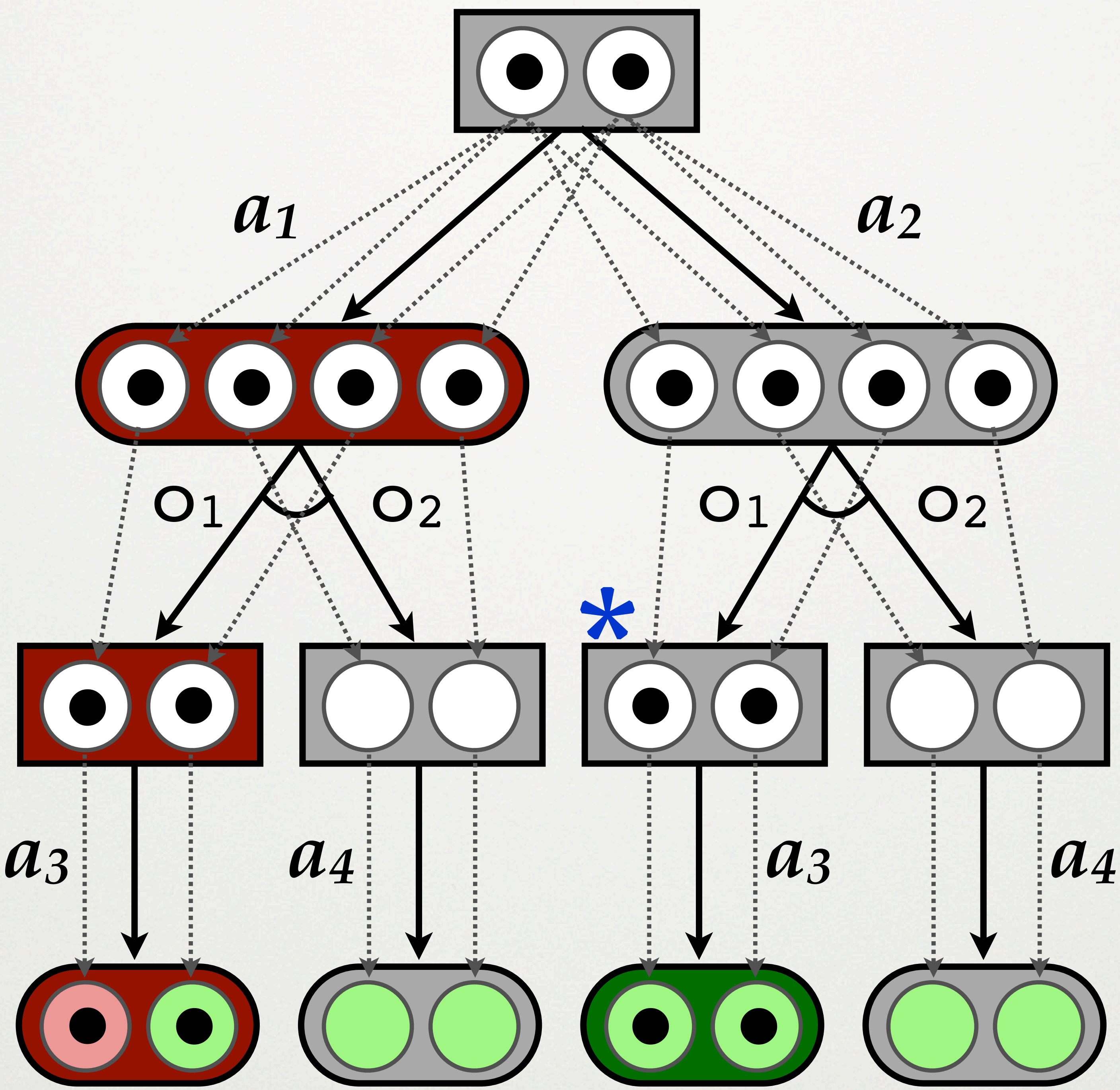
# DFS



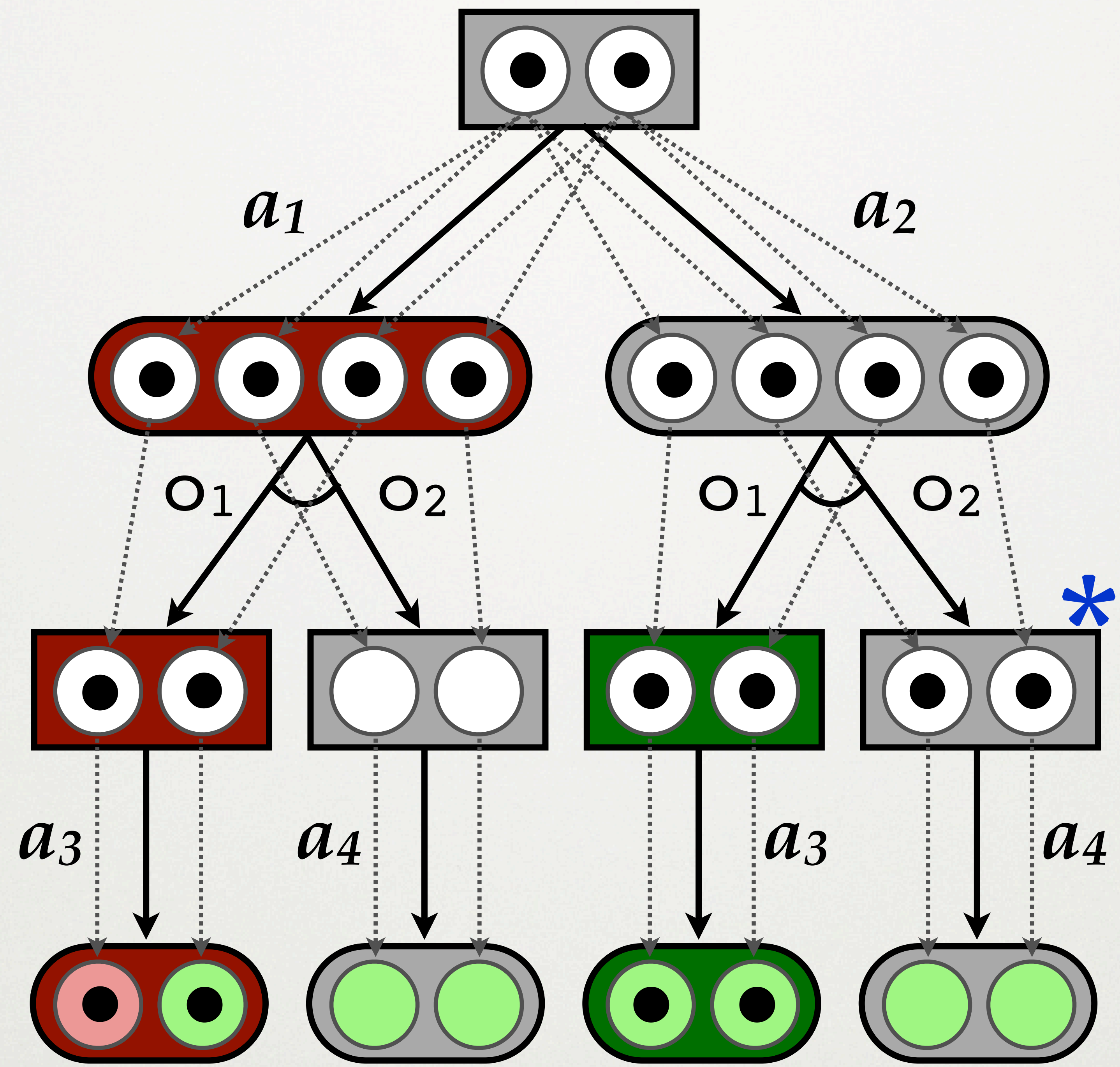
# DFS



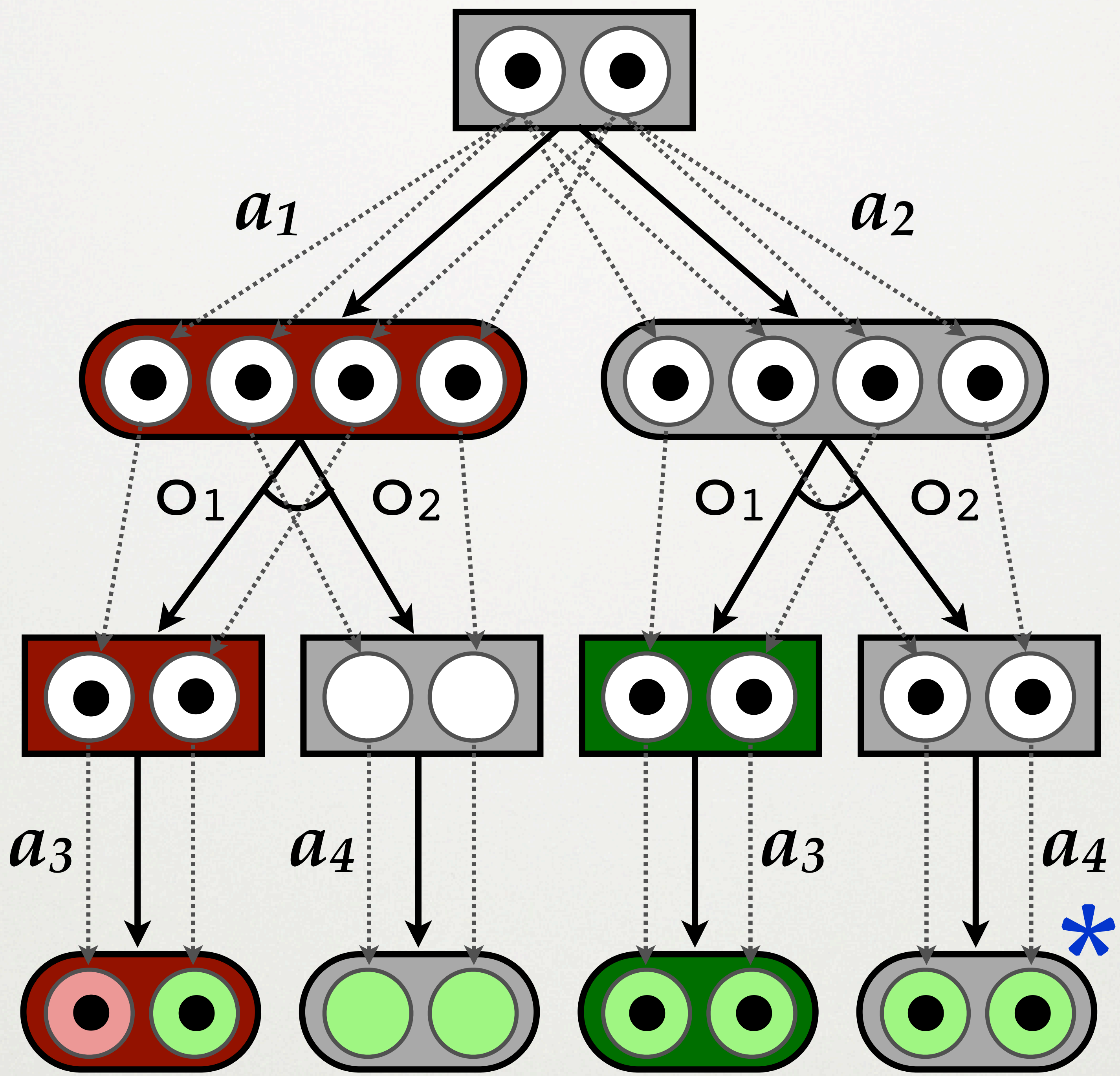
# DFS



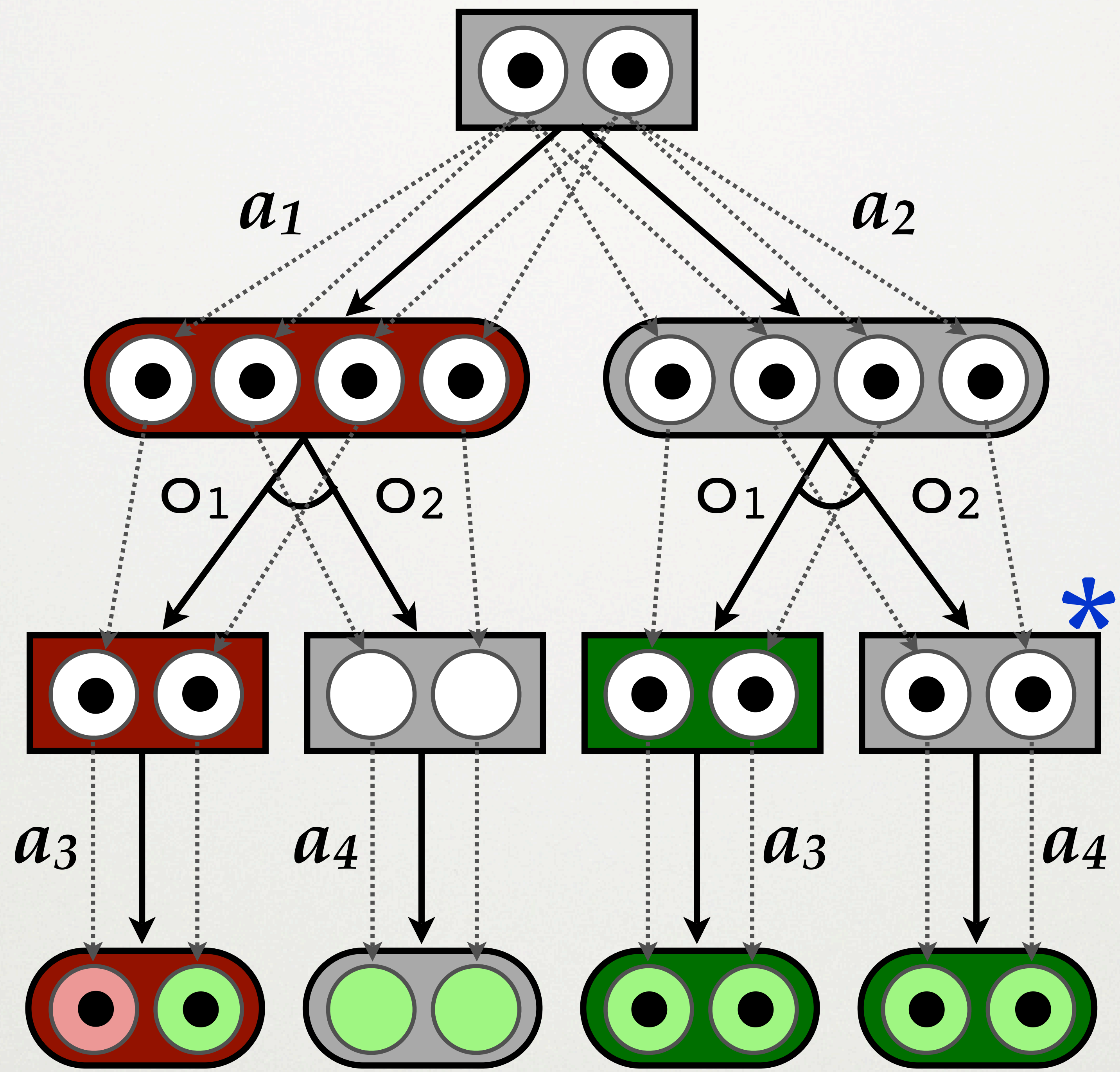
# DFS



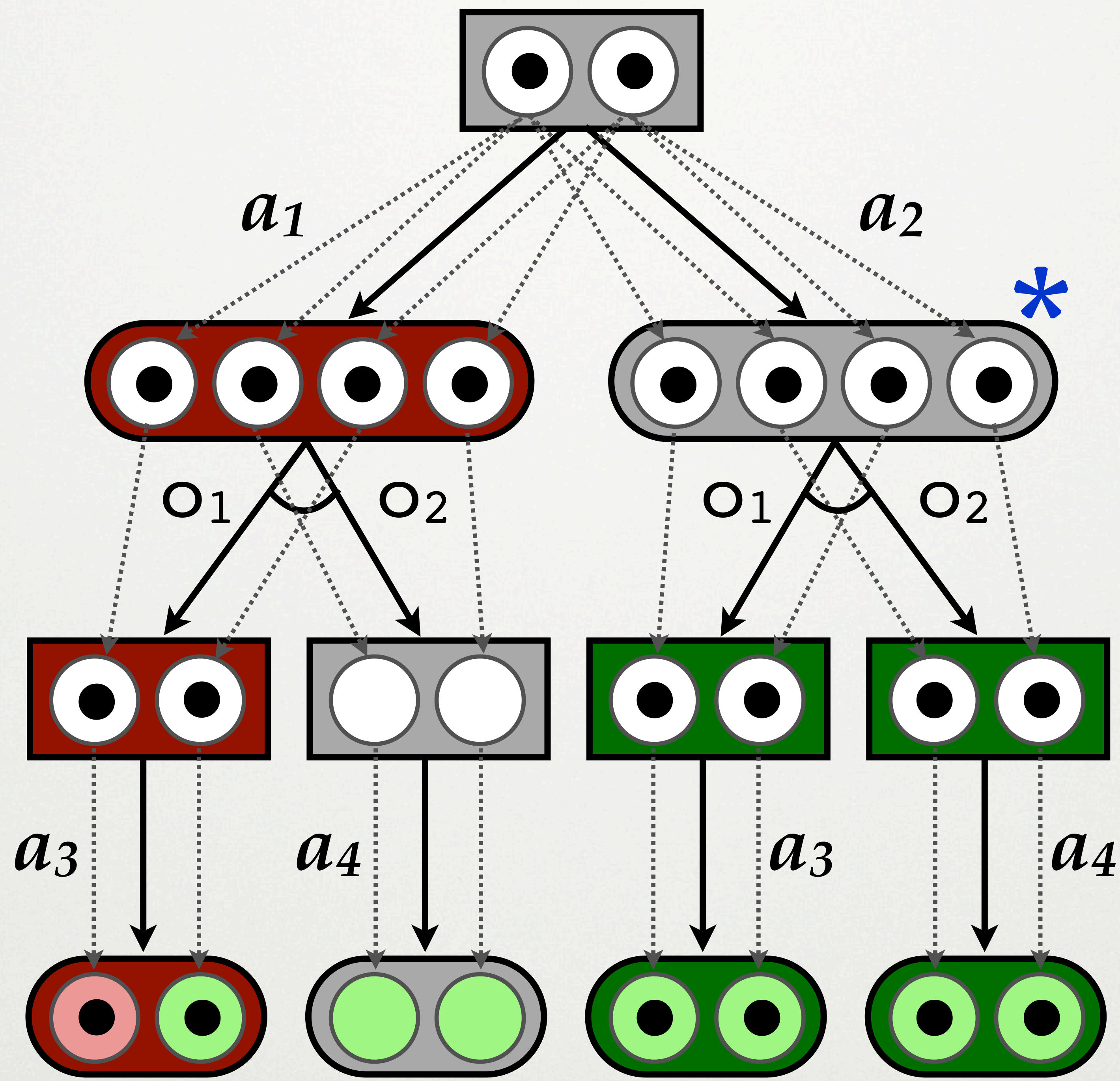
# DFS



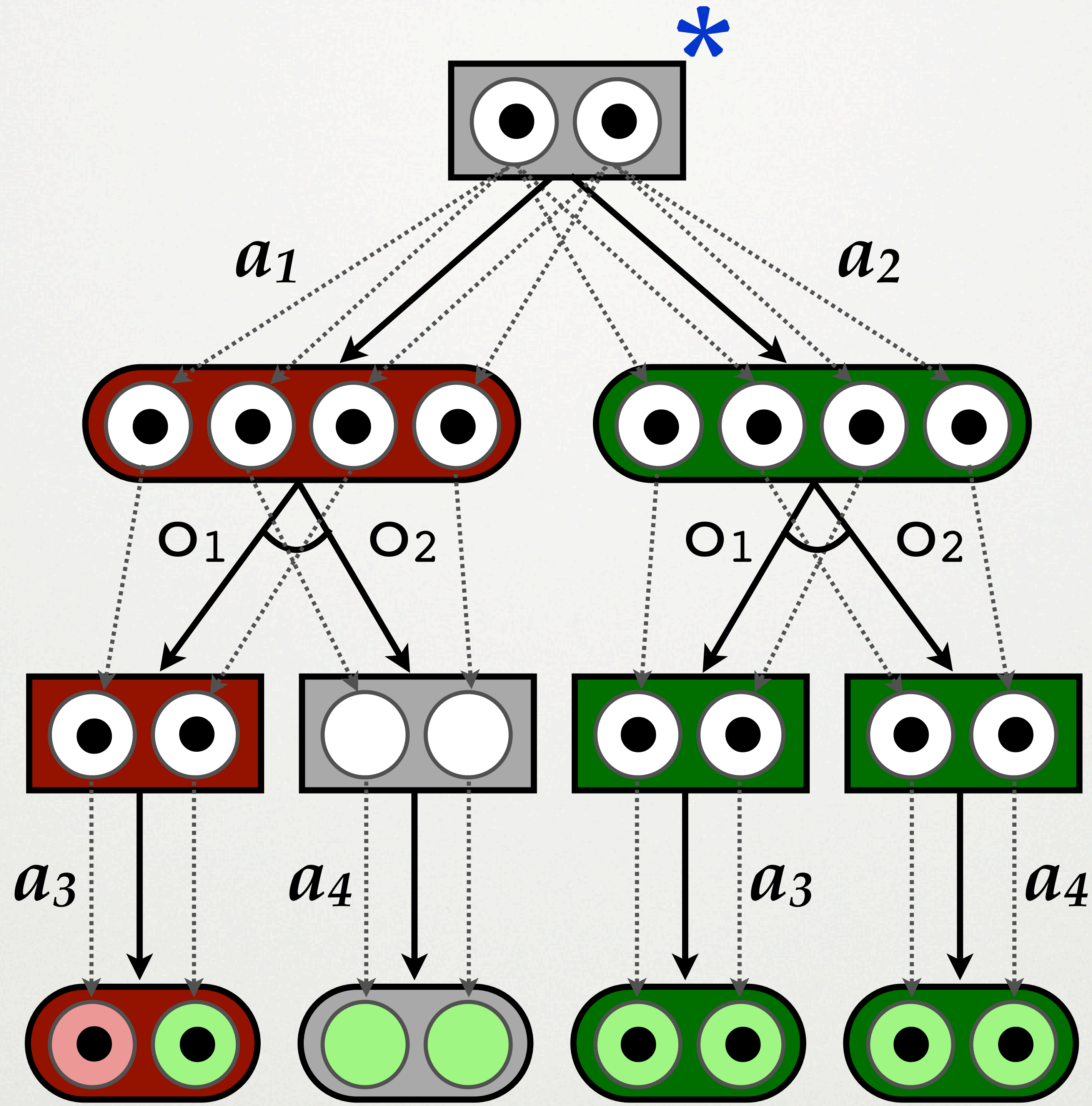
# DFS



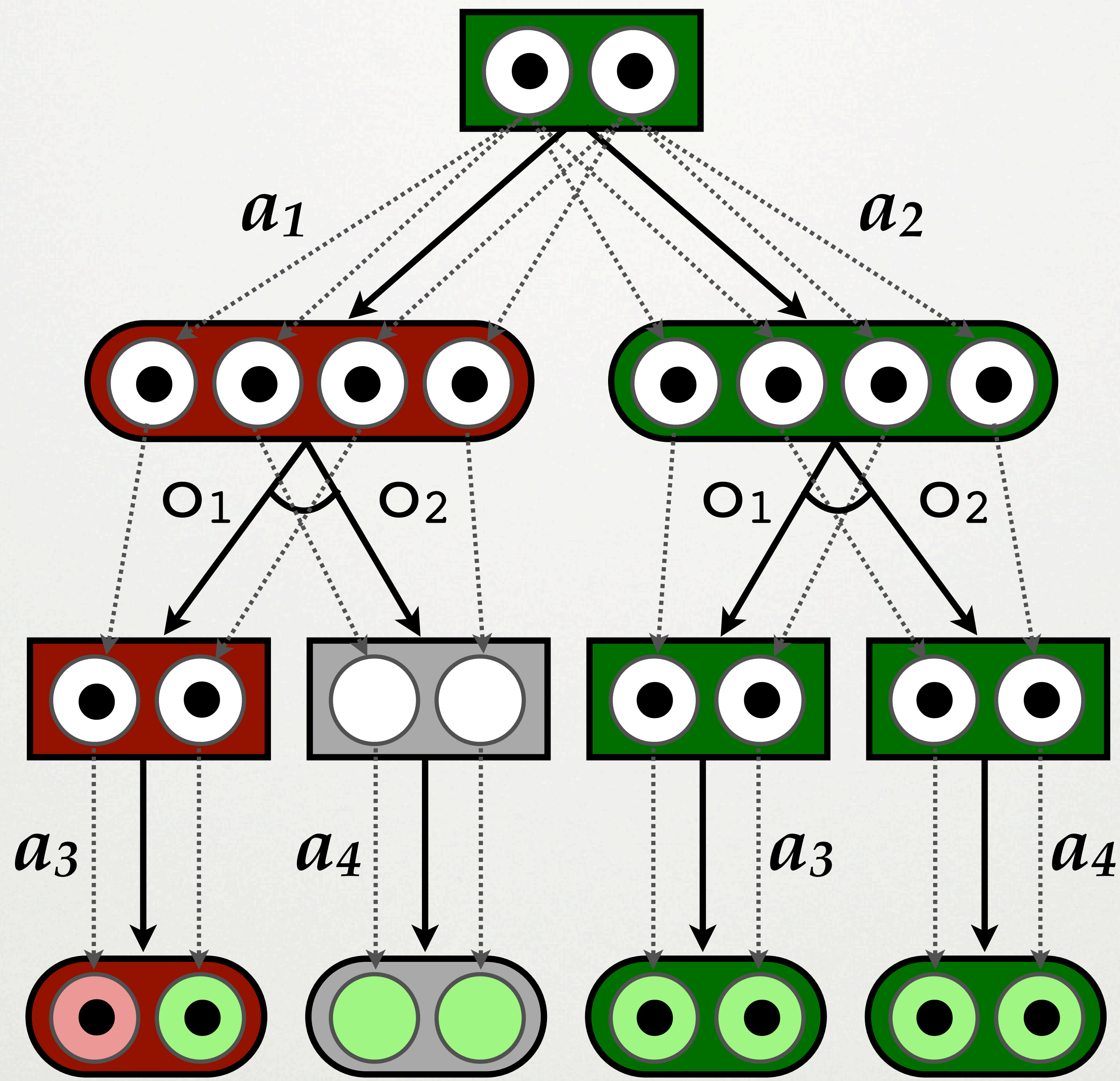
# DFS



# DFS



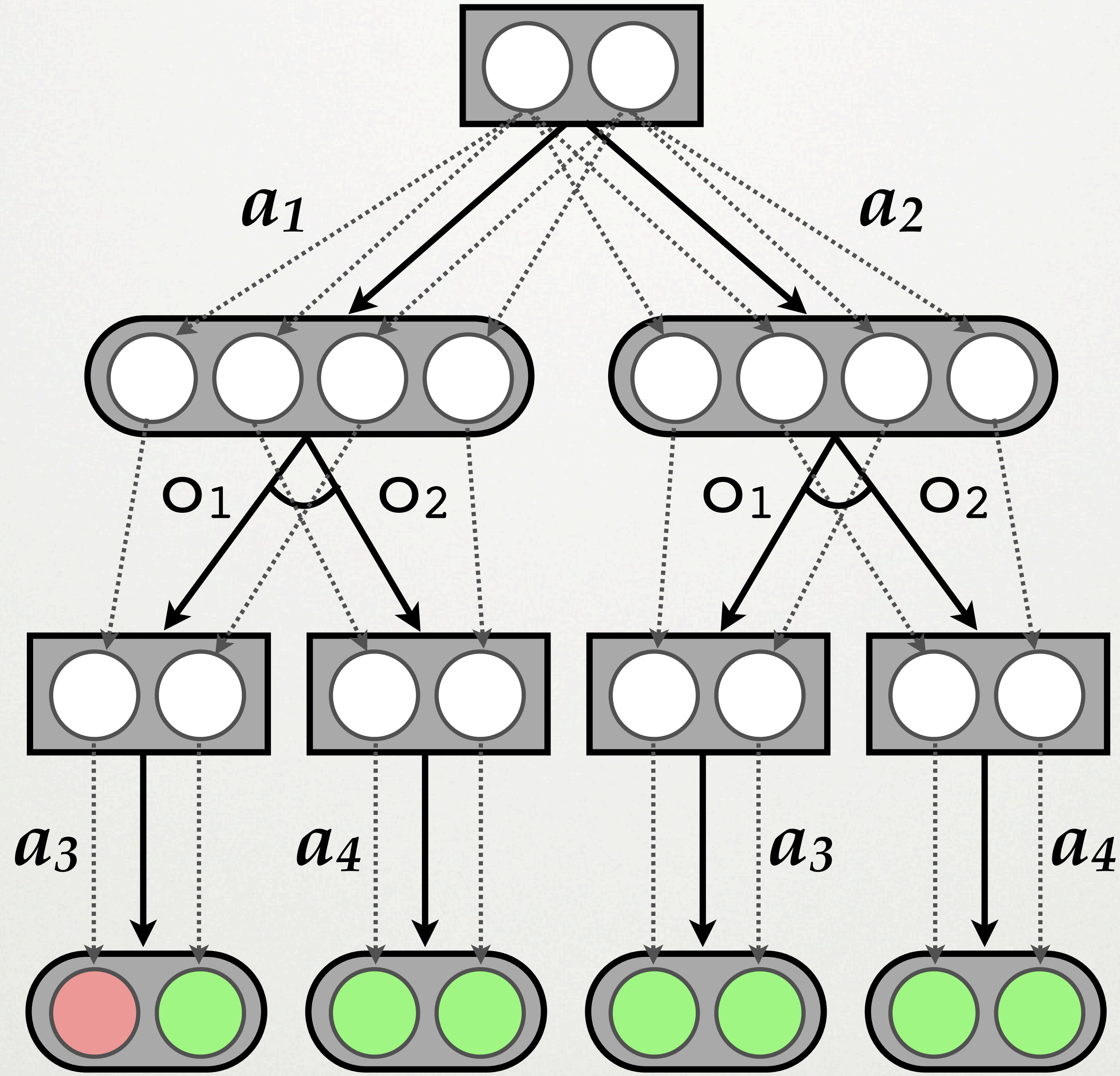
# DFS



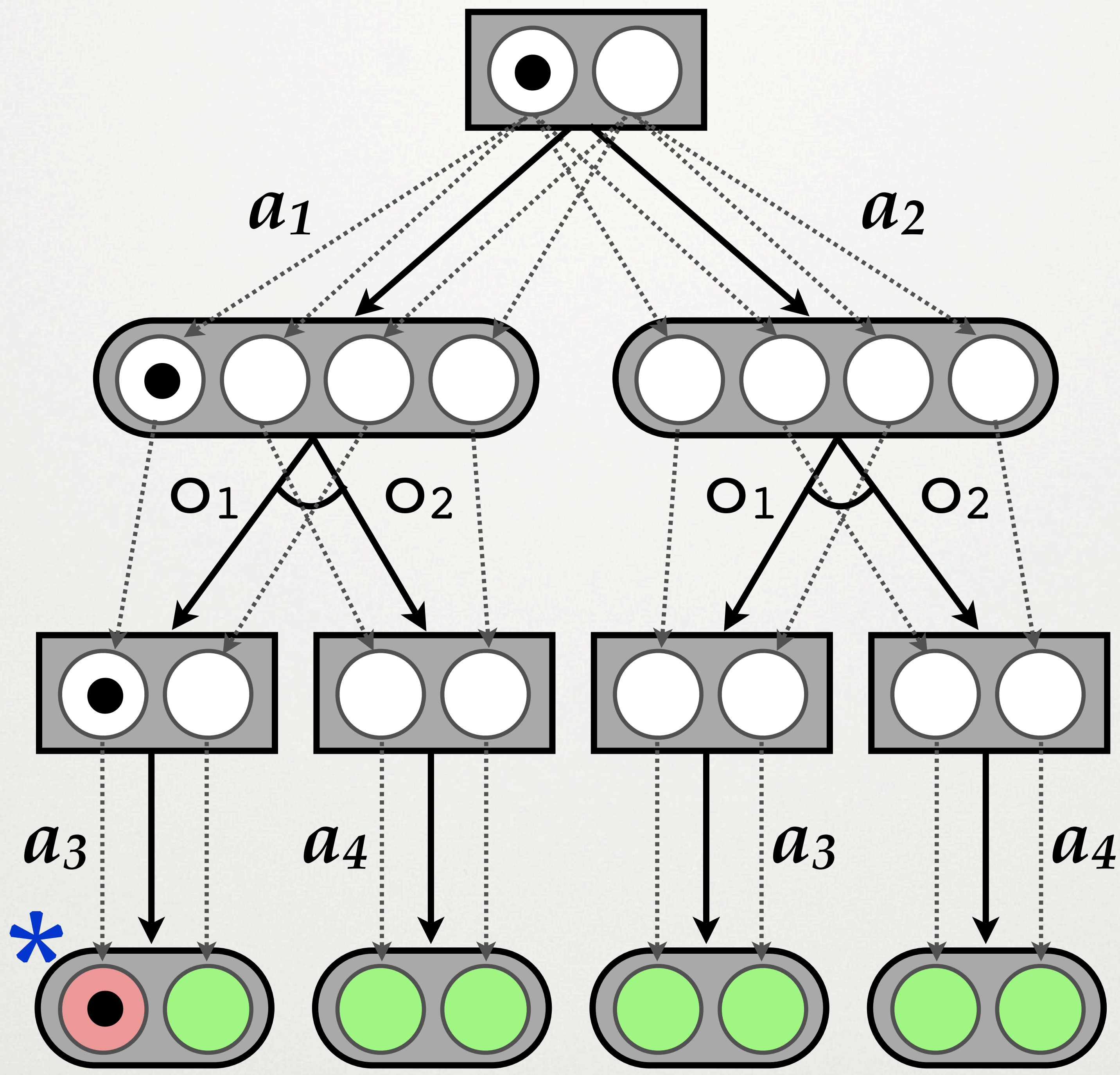
# Search Algorithm DBU

- **DBU** = depth, breadth, then uncertainty
- Builds proofs one physical state at a time
  - fail fast if subset found unsolvable; enables minimal disproofs
  - can be significantly faster than **DFS** [Russell + Wolfe (2005)]

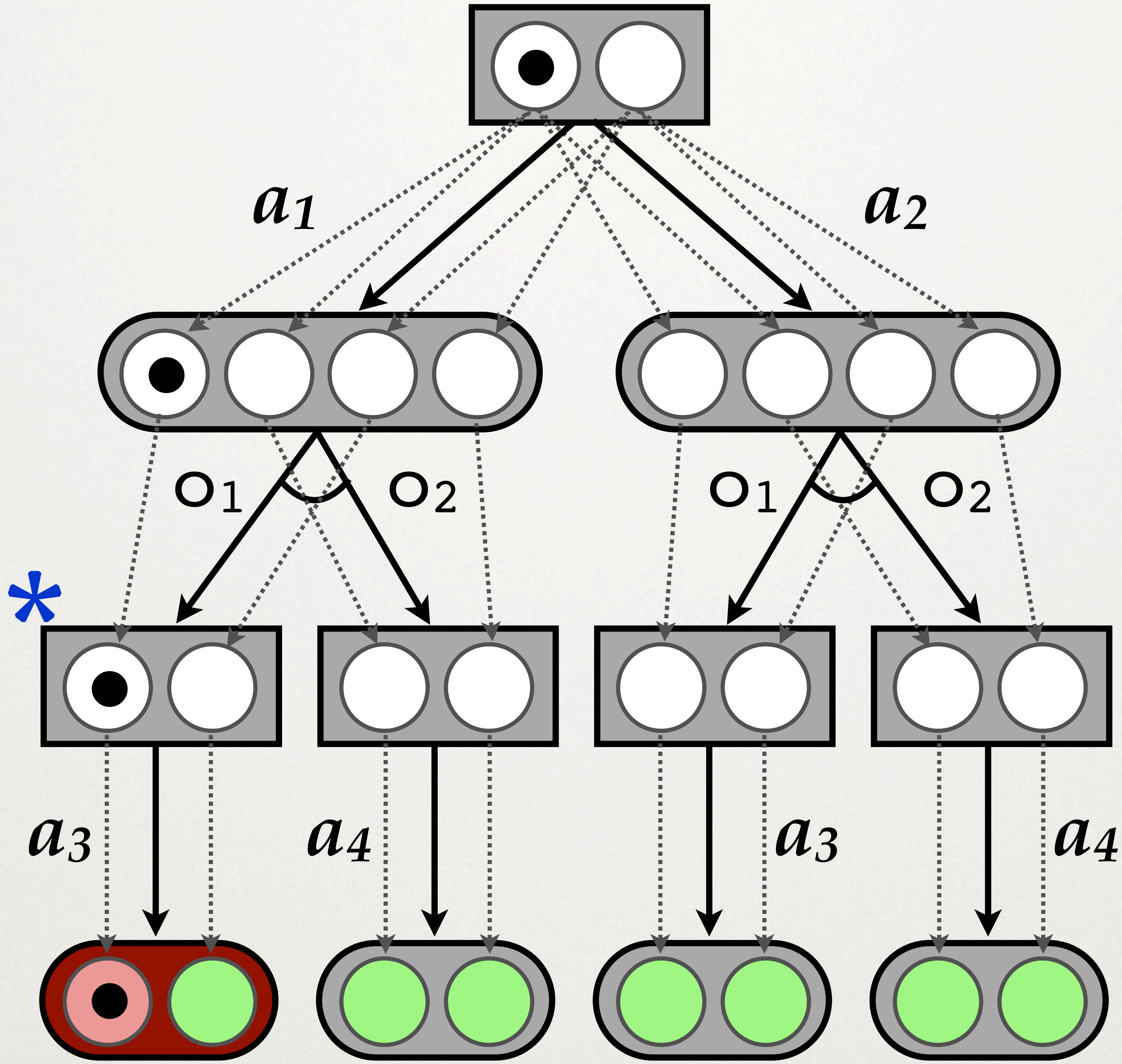
# DBU



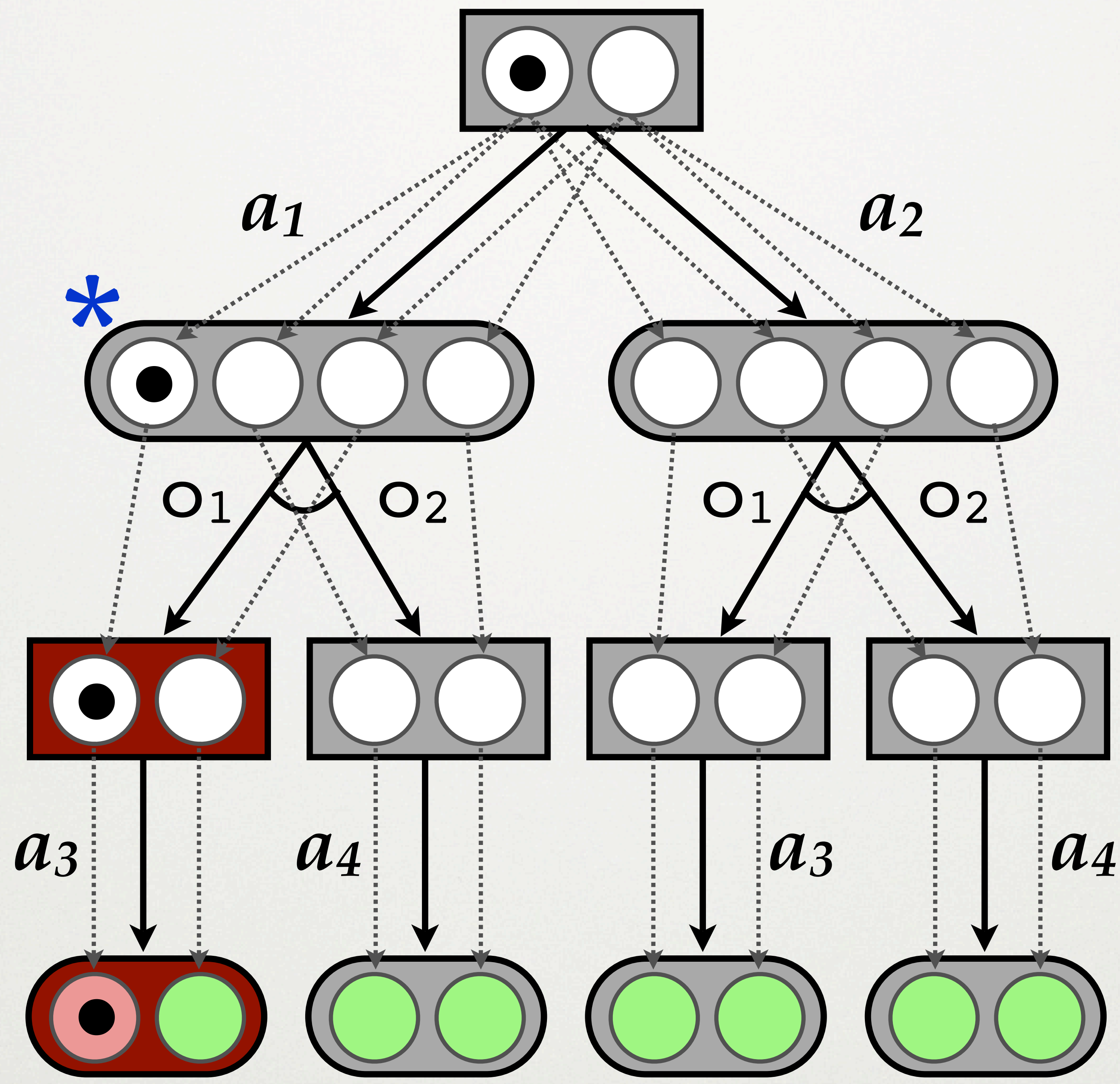
# DBU



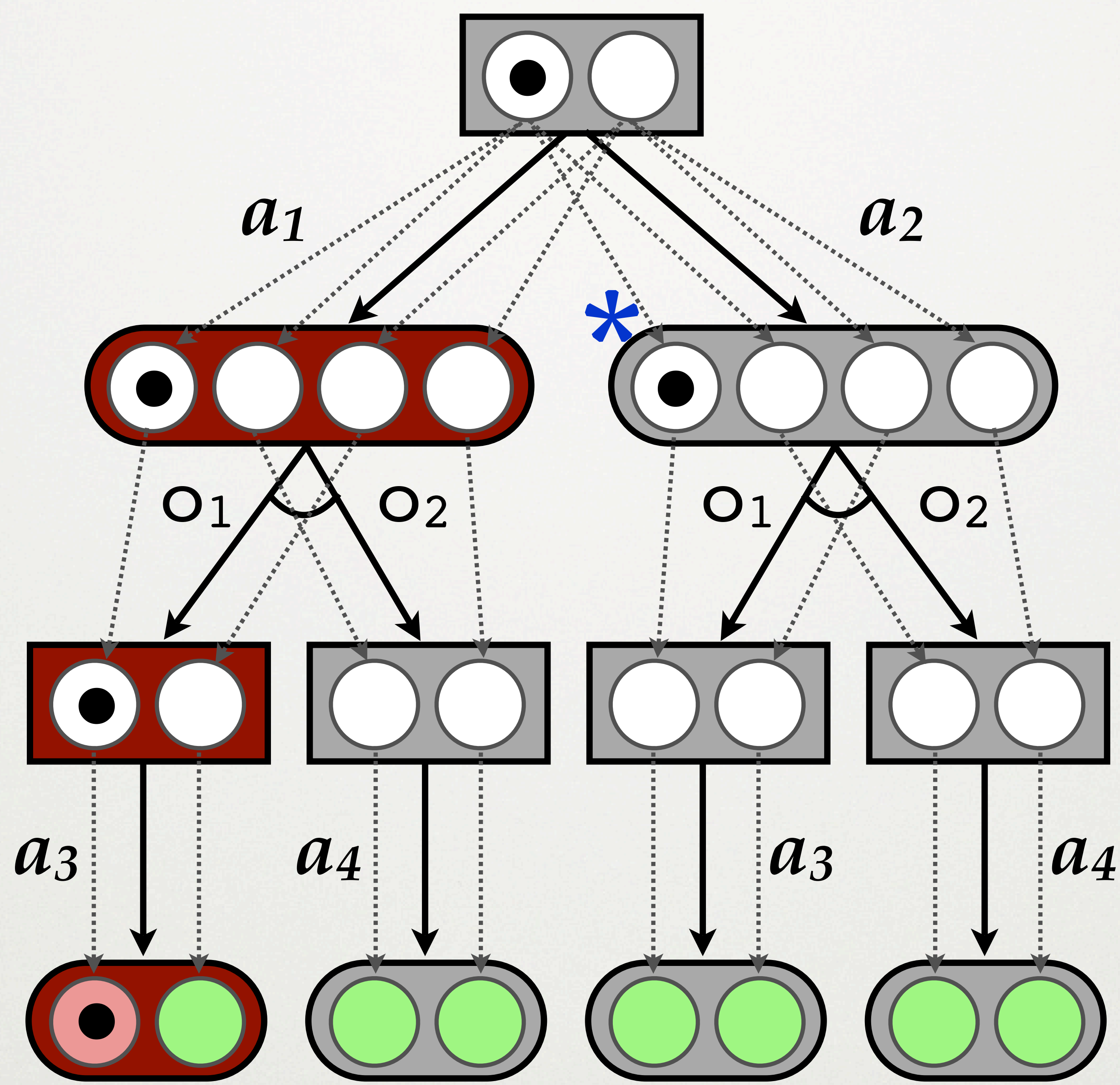
# DBU



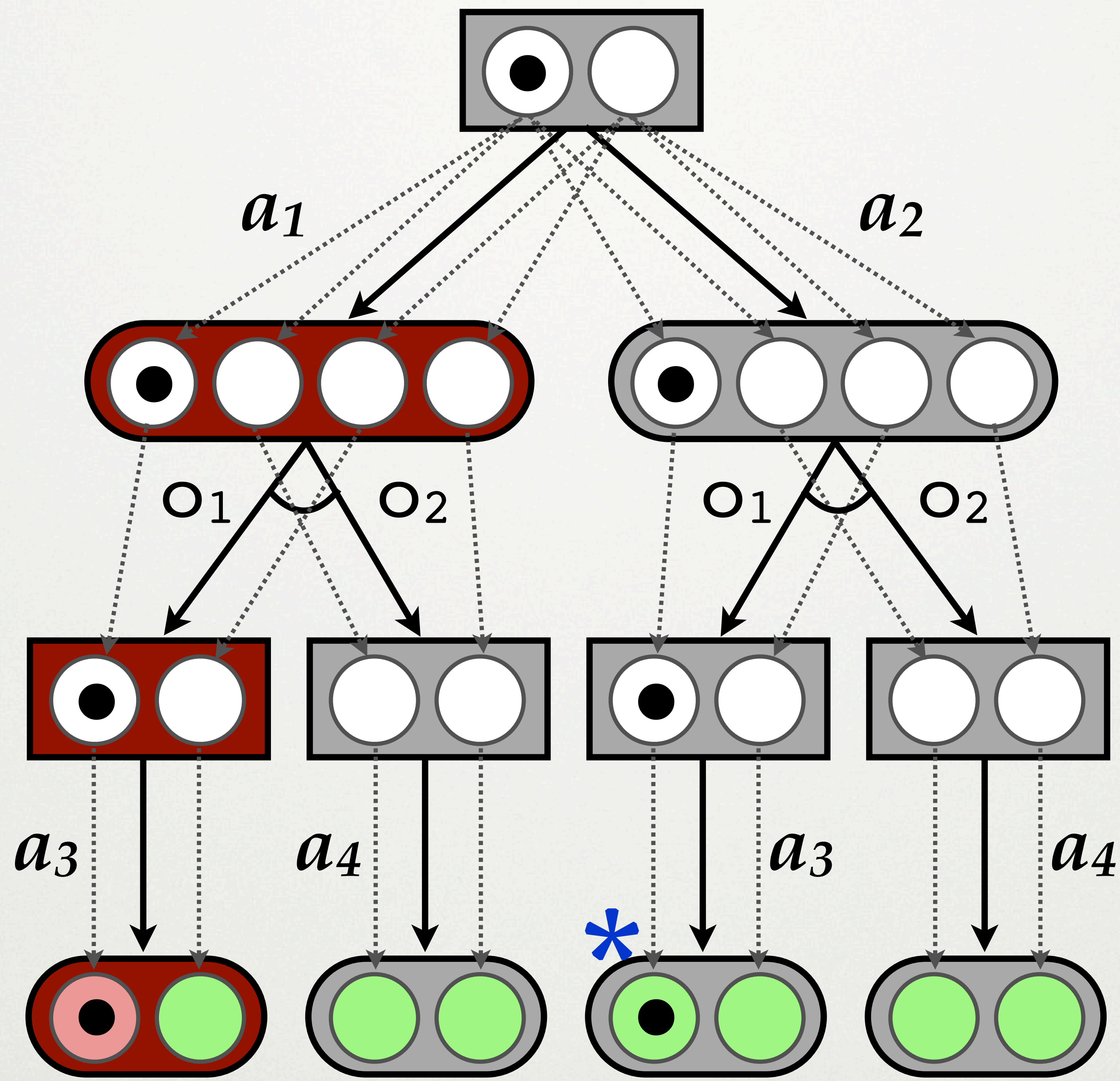
# DBU



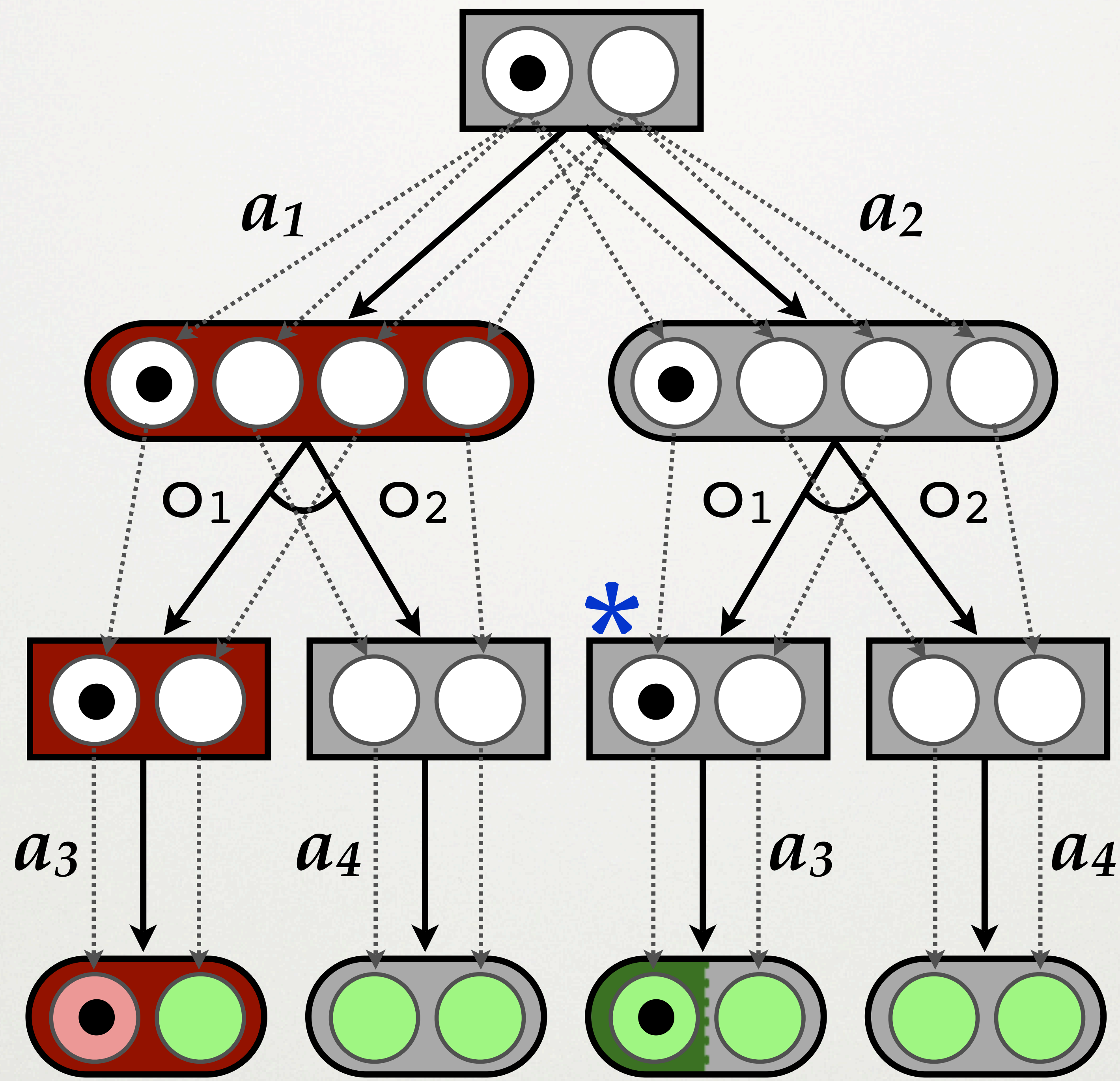
# DBU



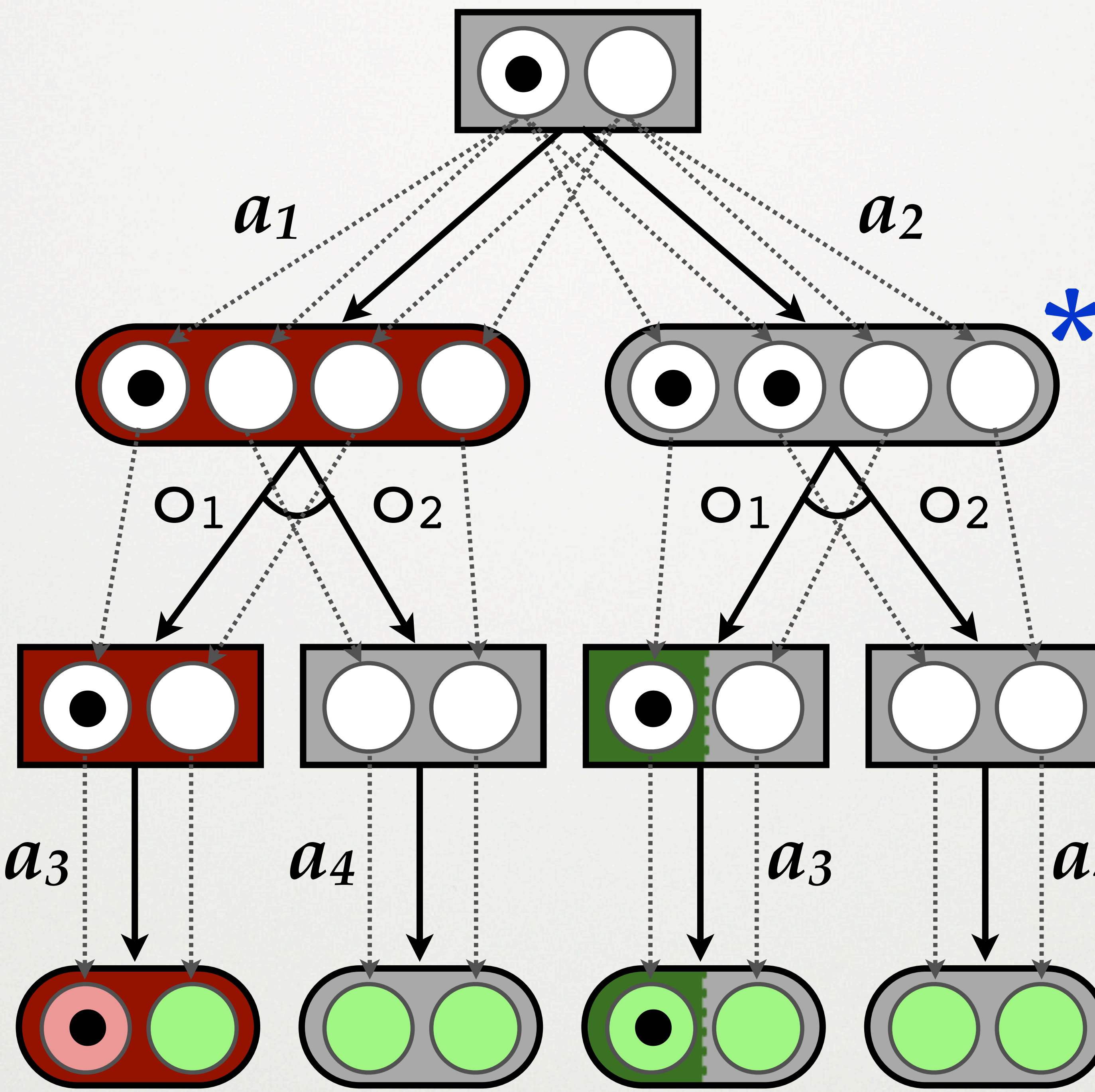
# DBU



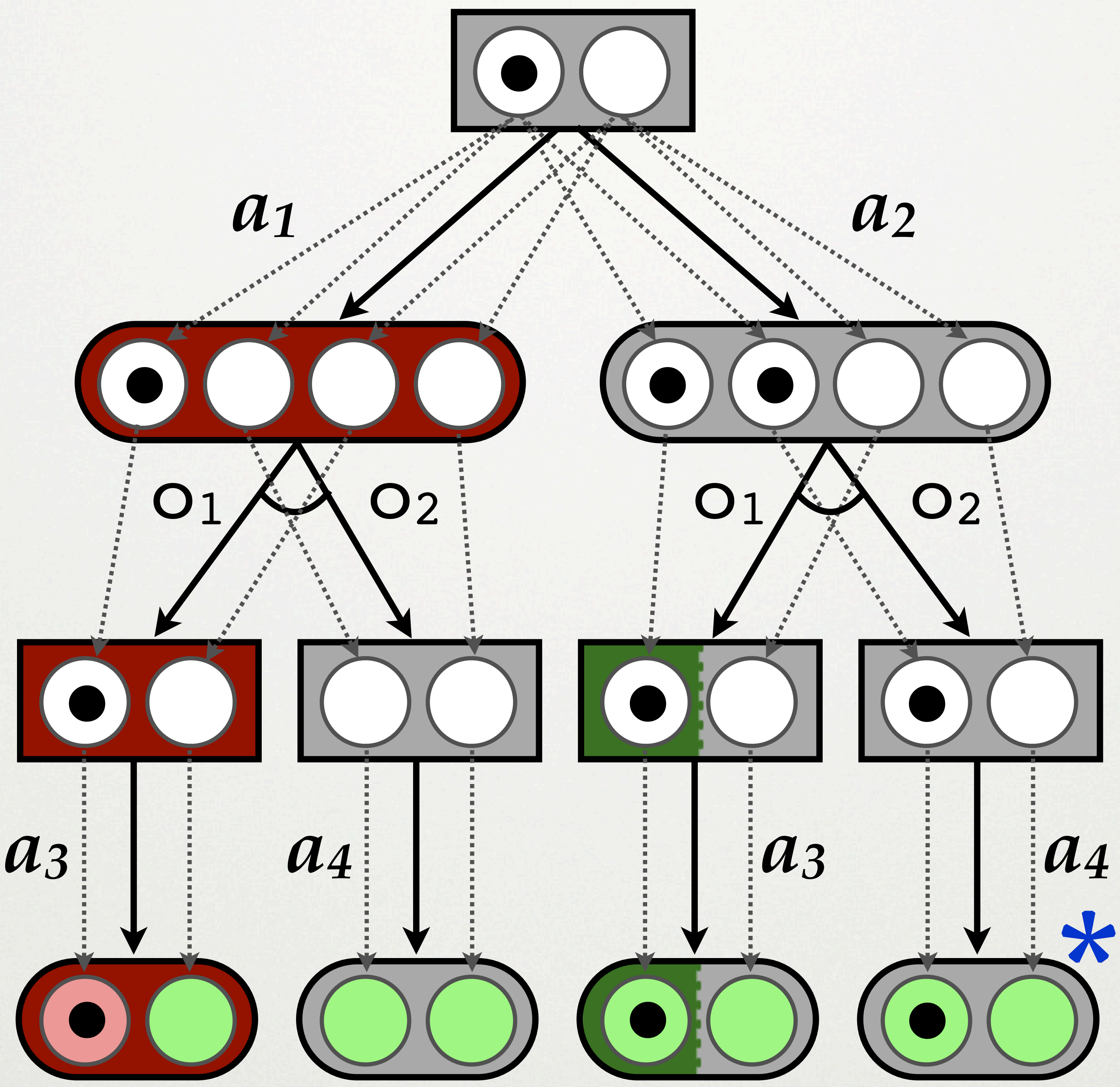
# DBU



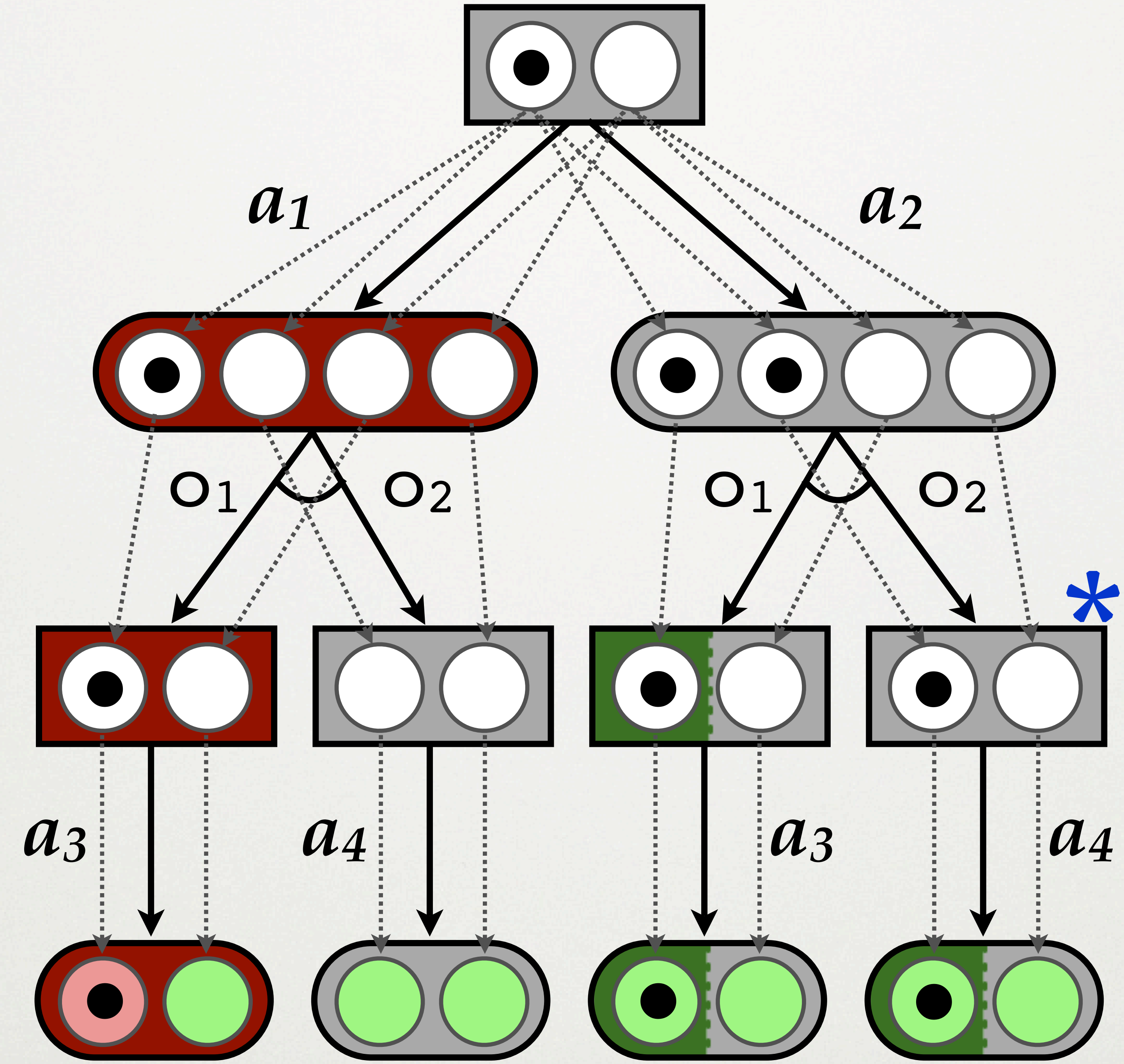
# DBU



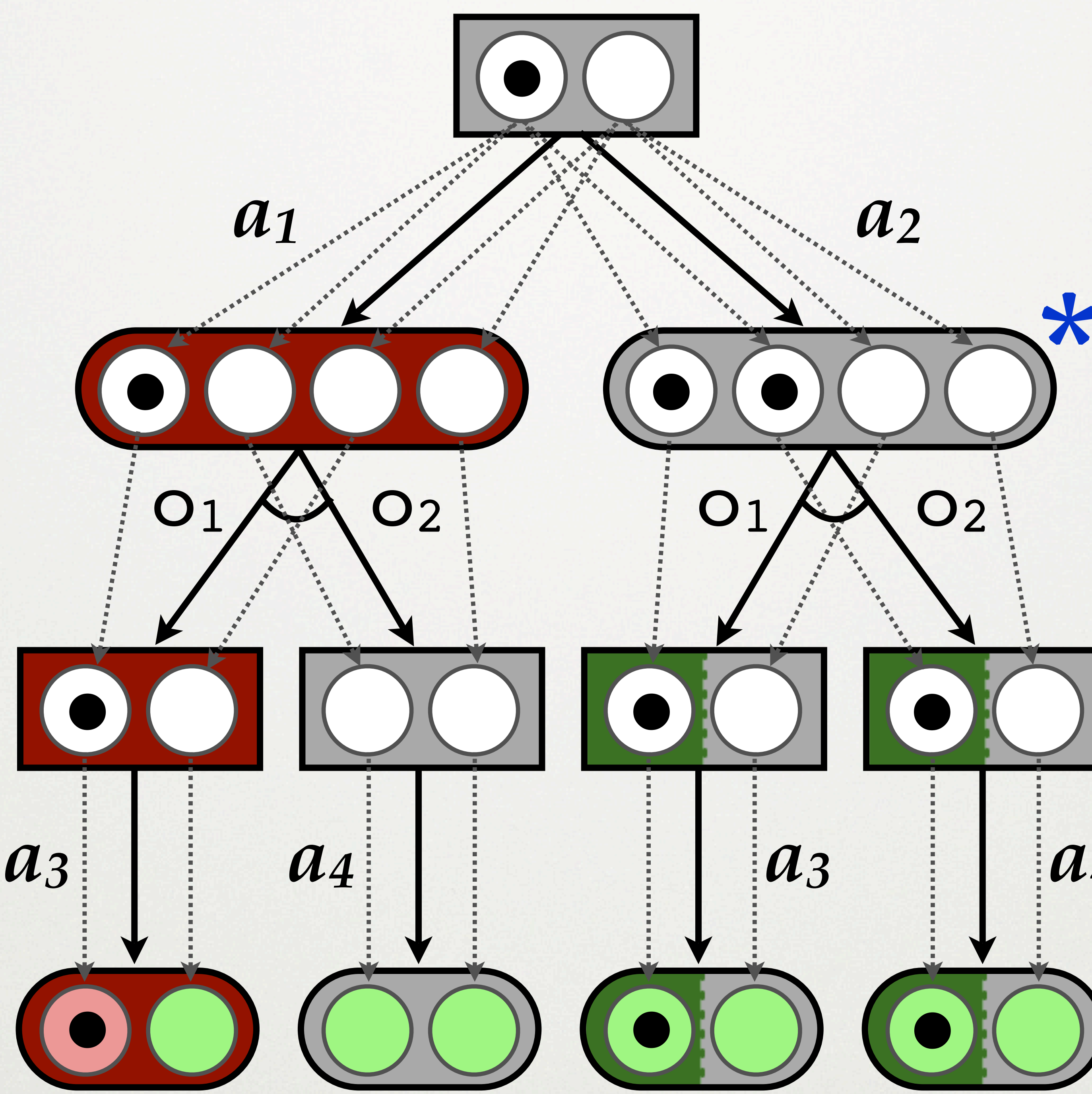
# DBU



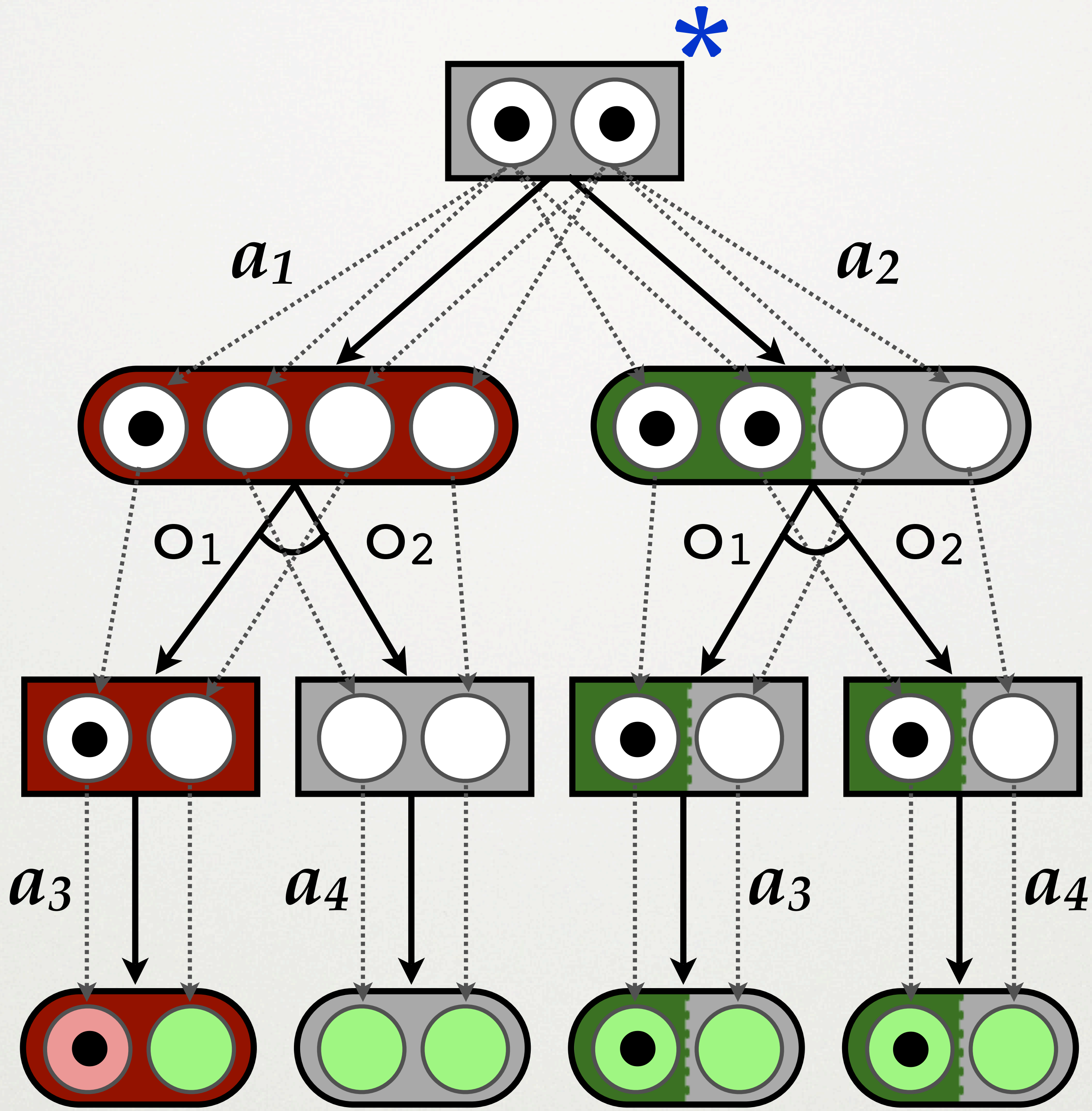
# DBU



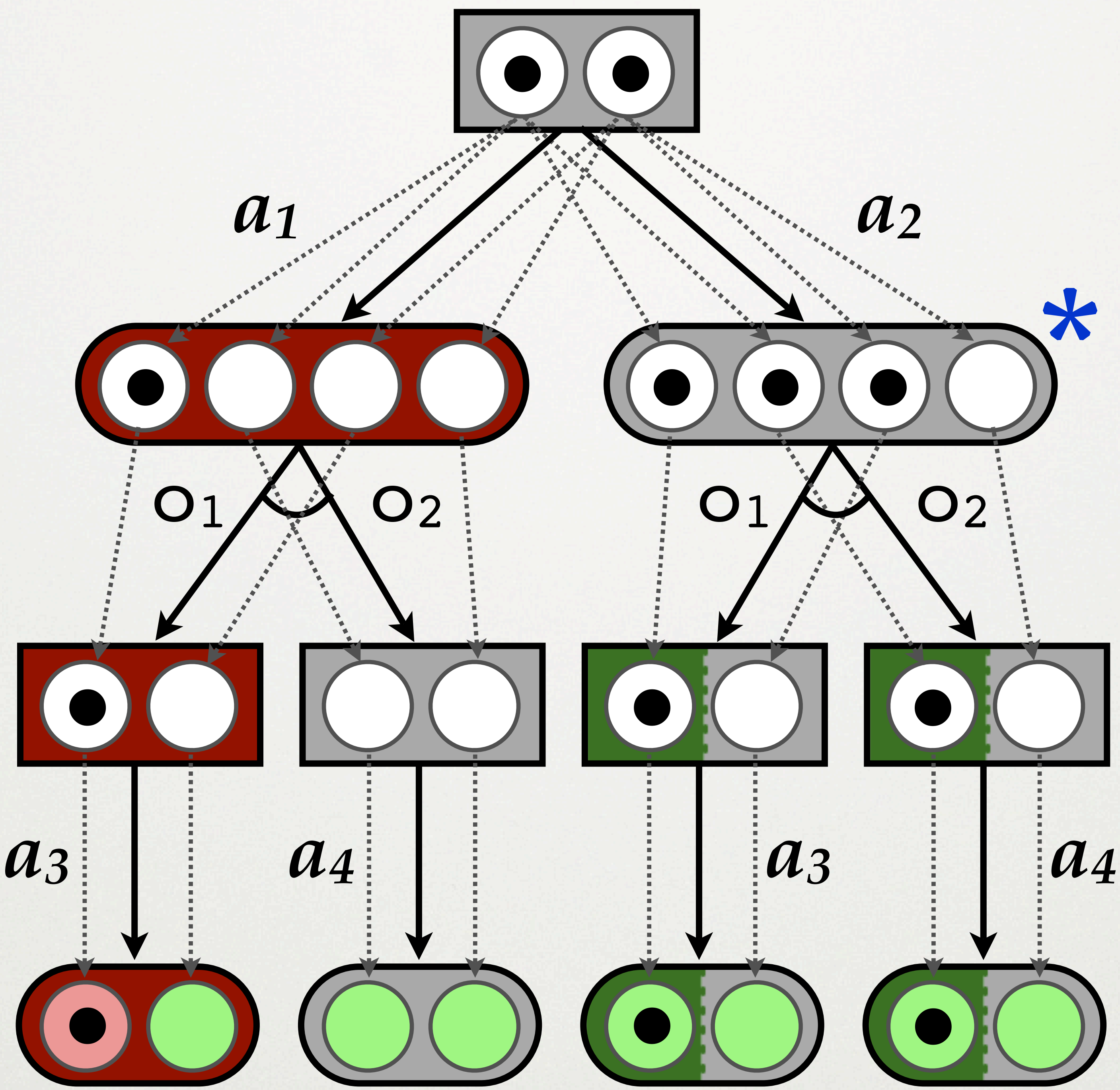
# DBU



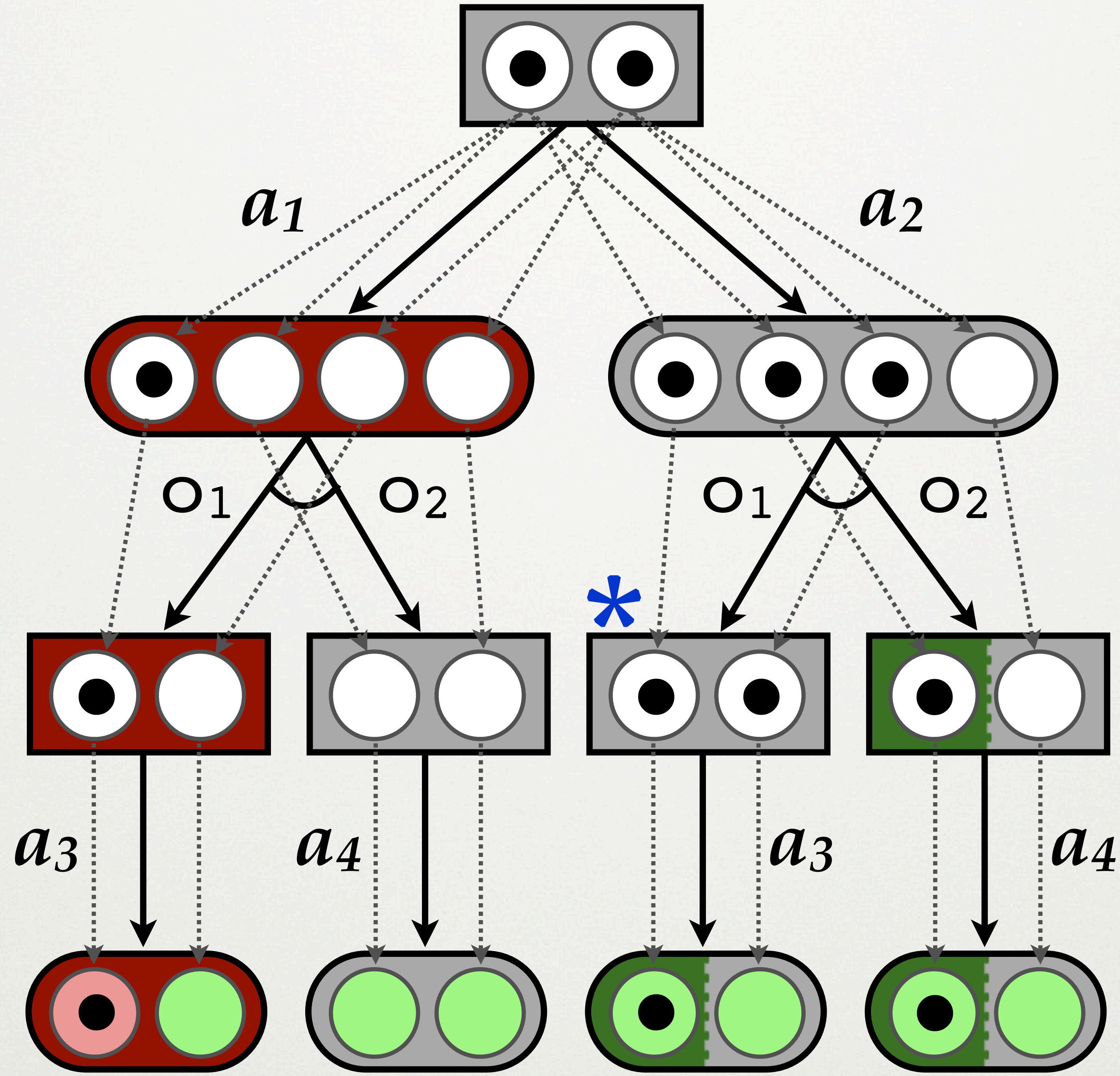
# DBU



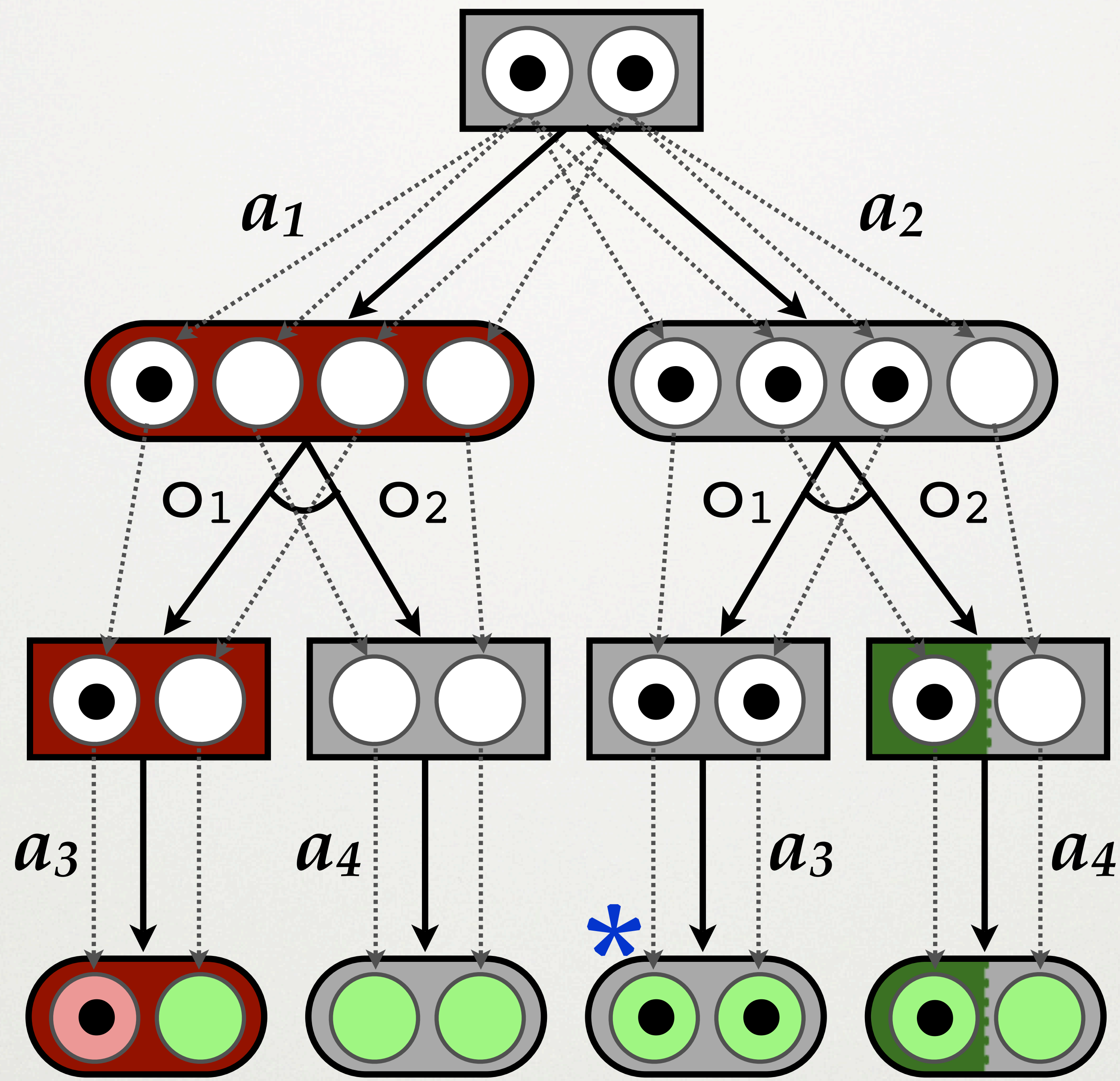
# DBU



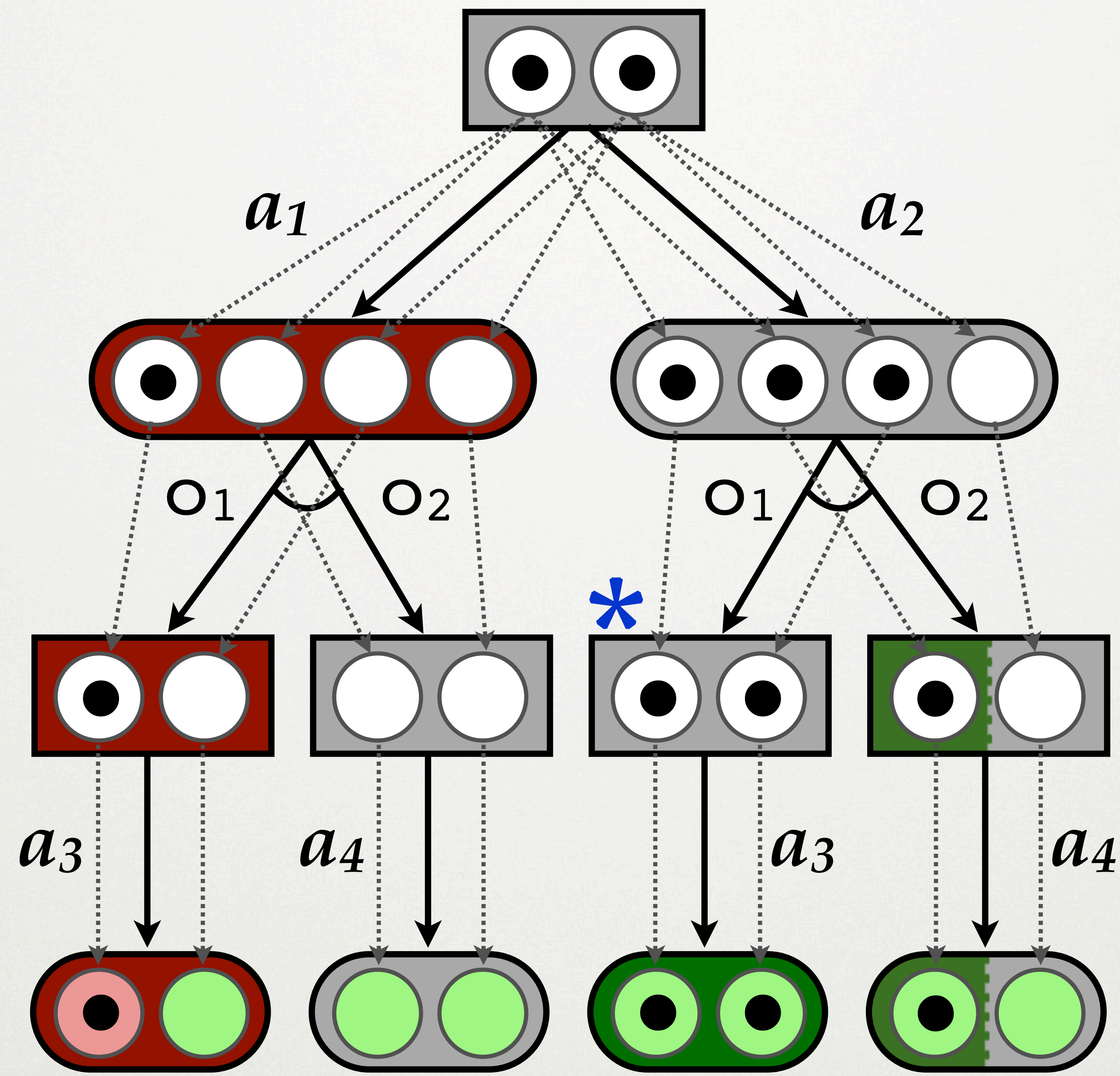
# DBU



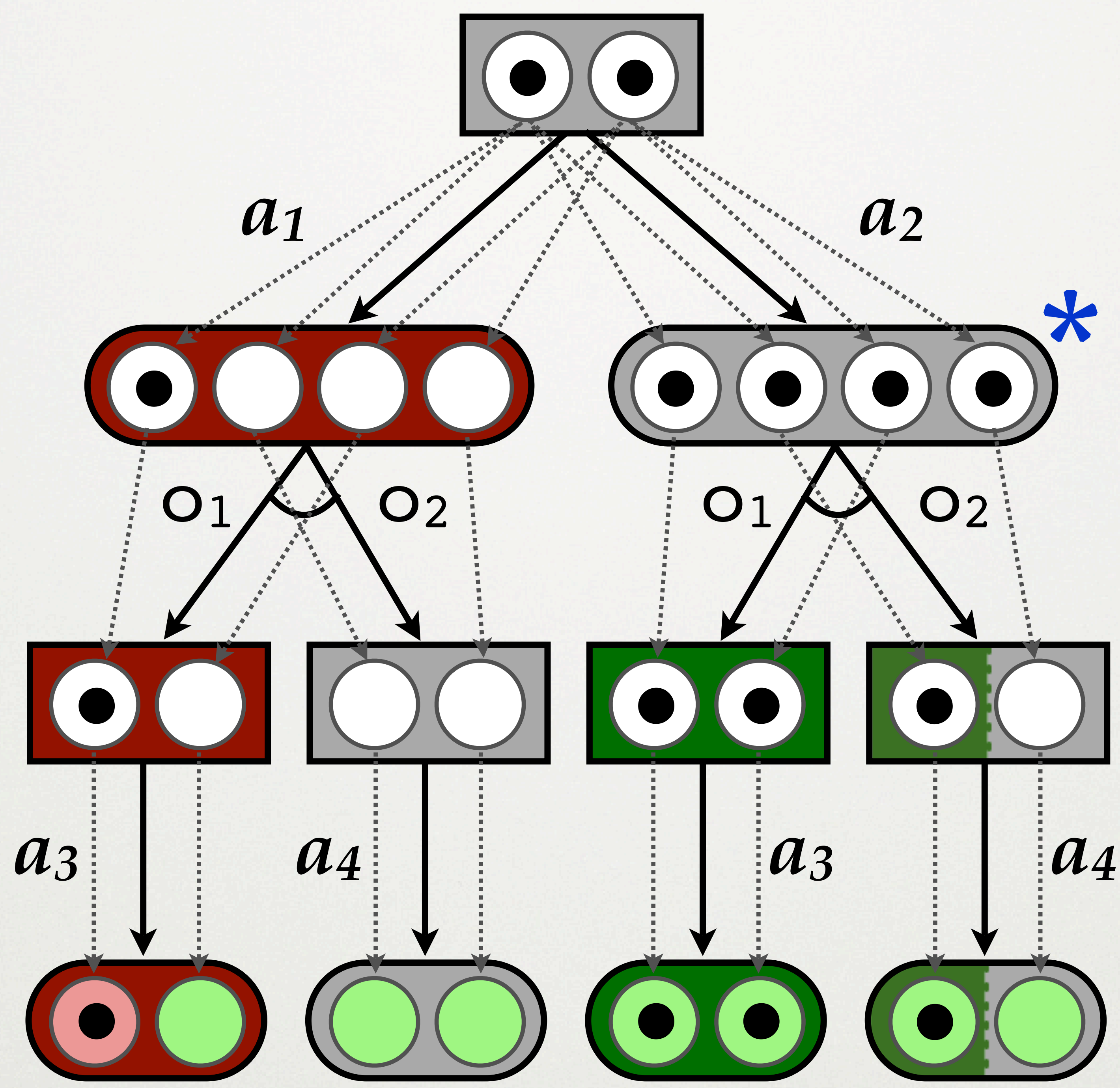
# DBU



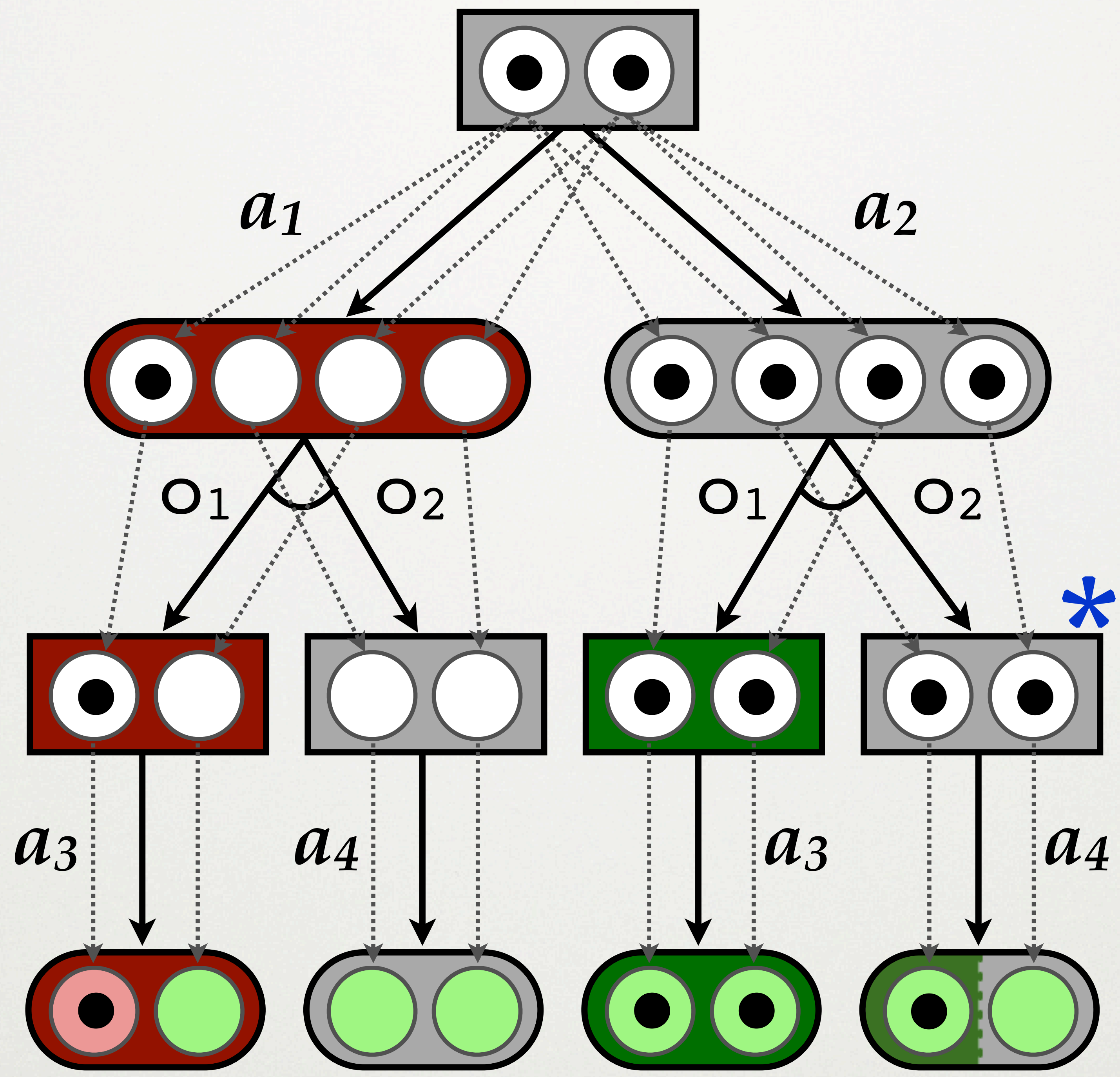
# DBU



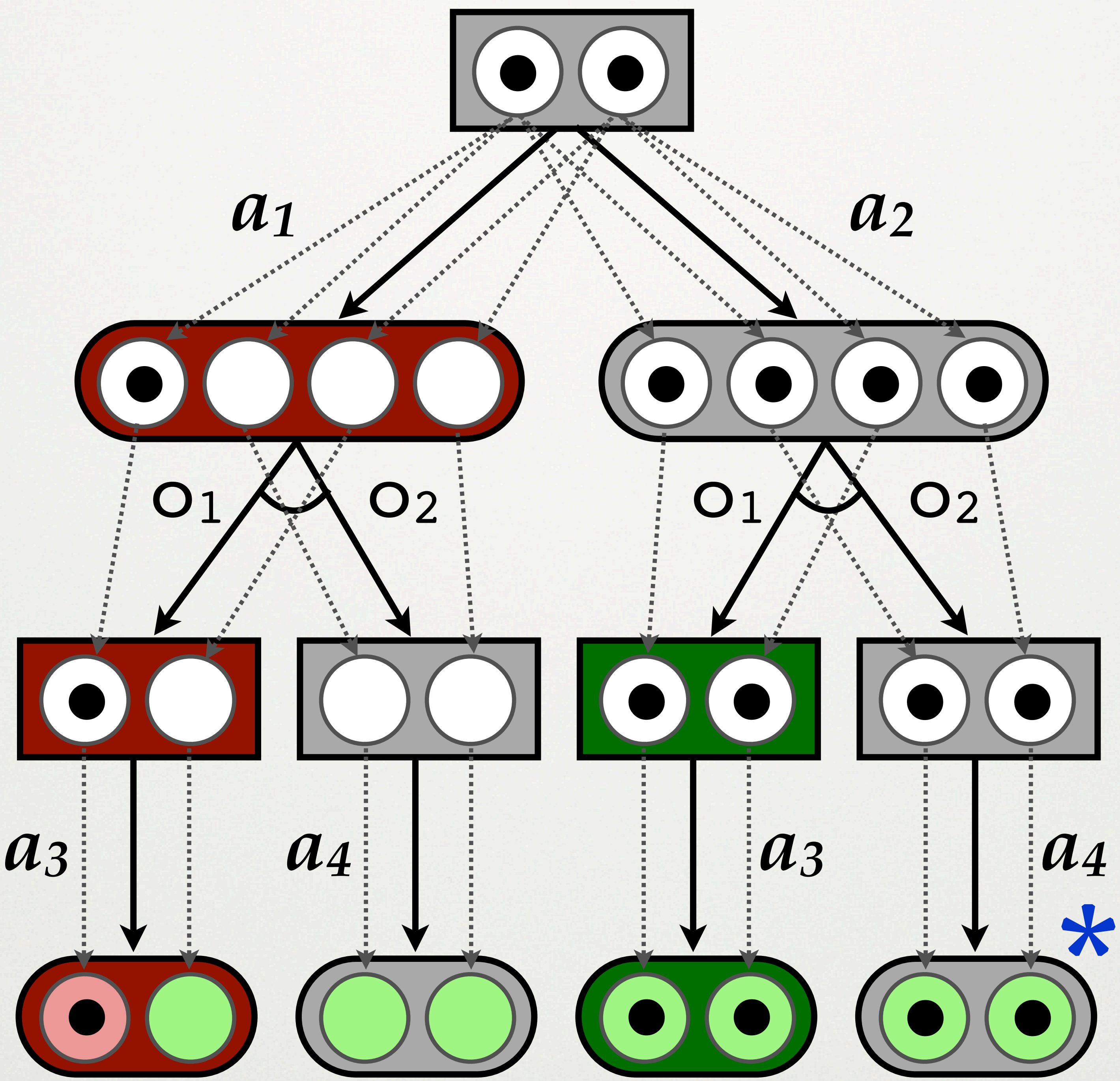
# DBU



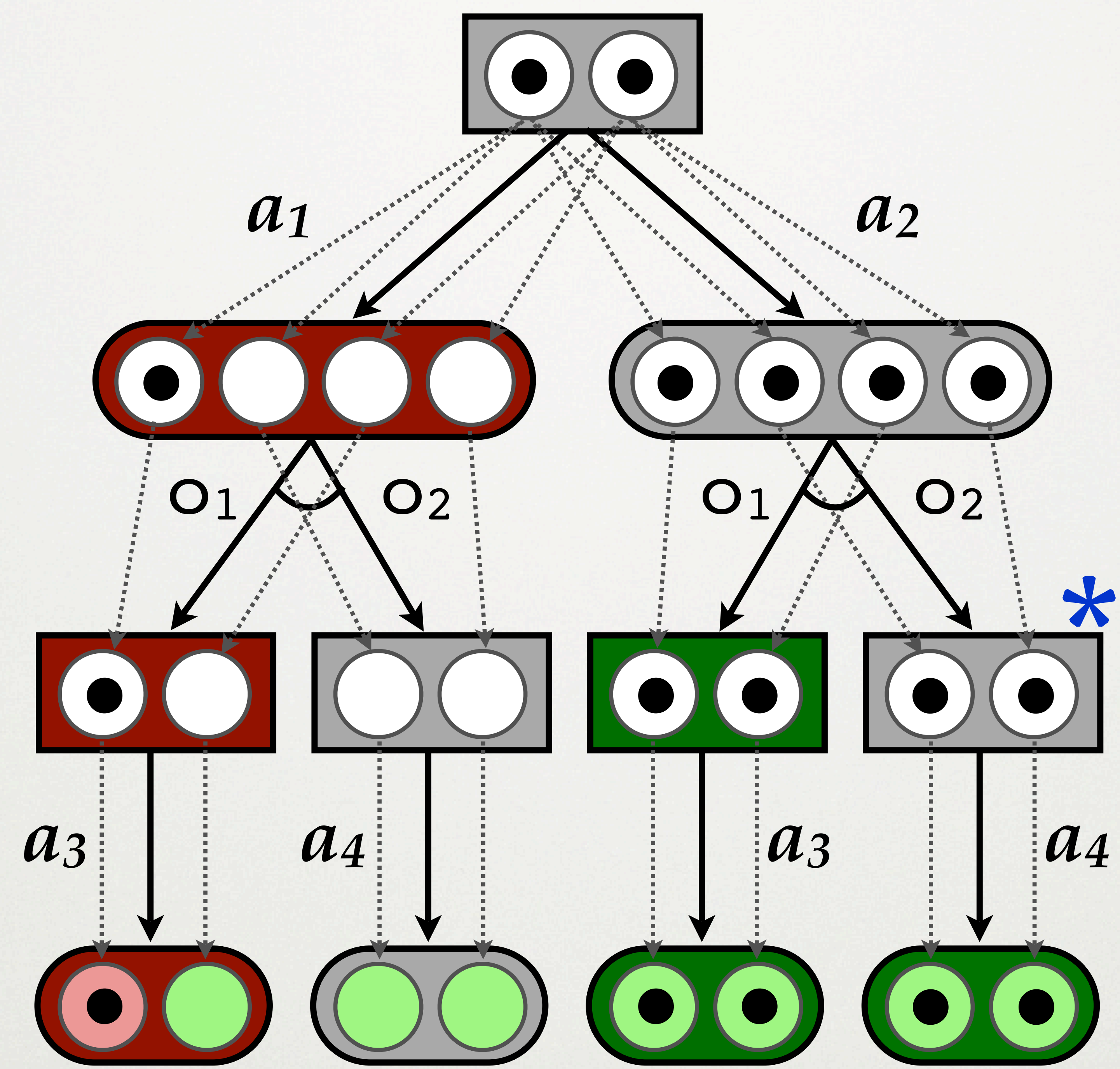
# DBU



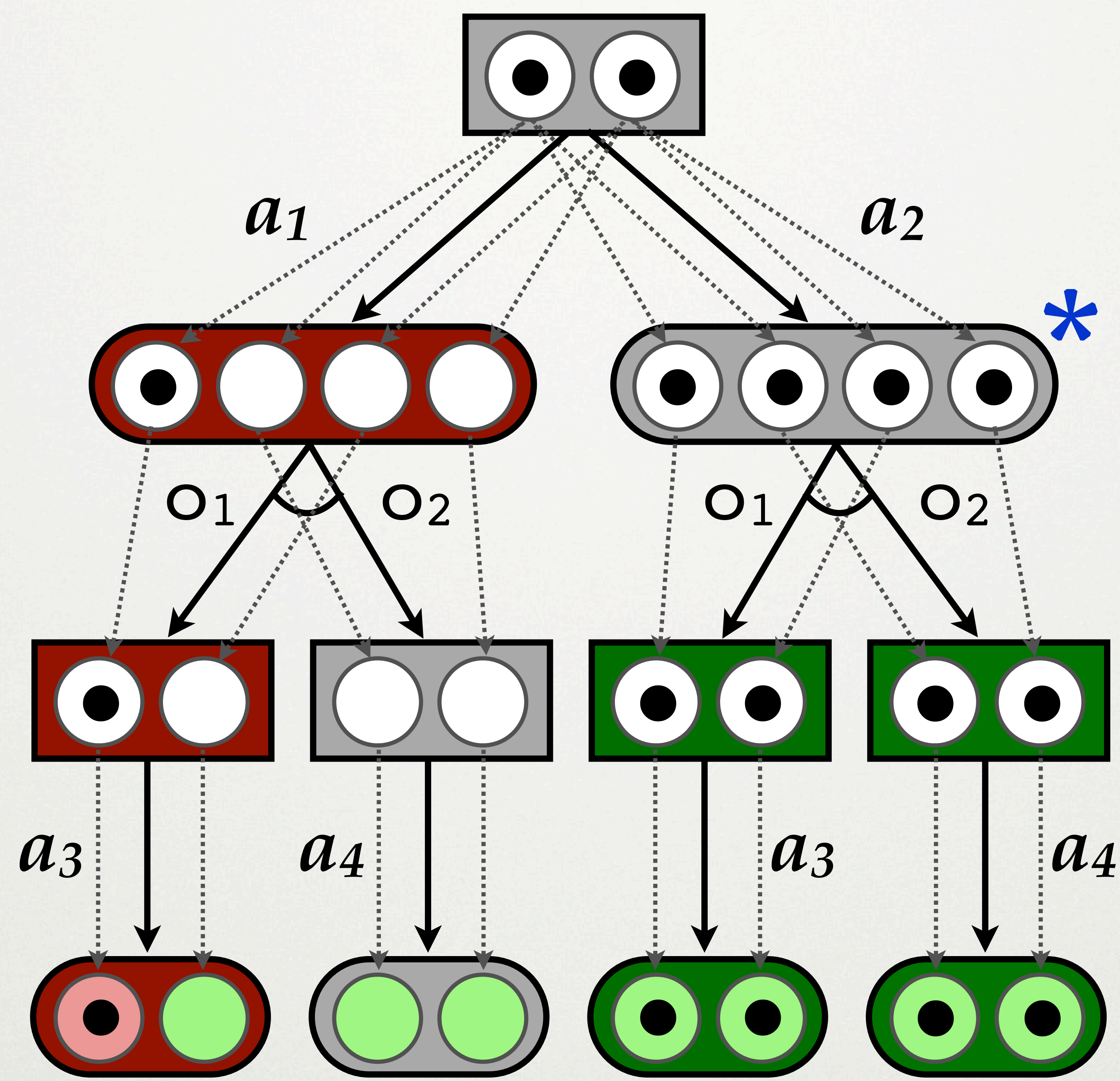
# DBU



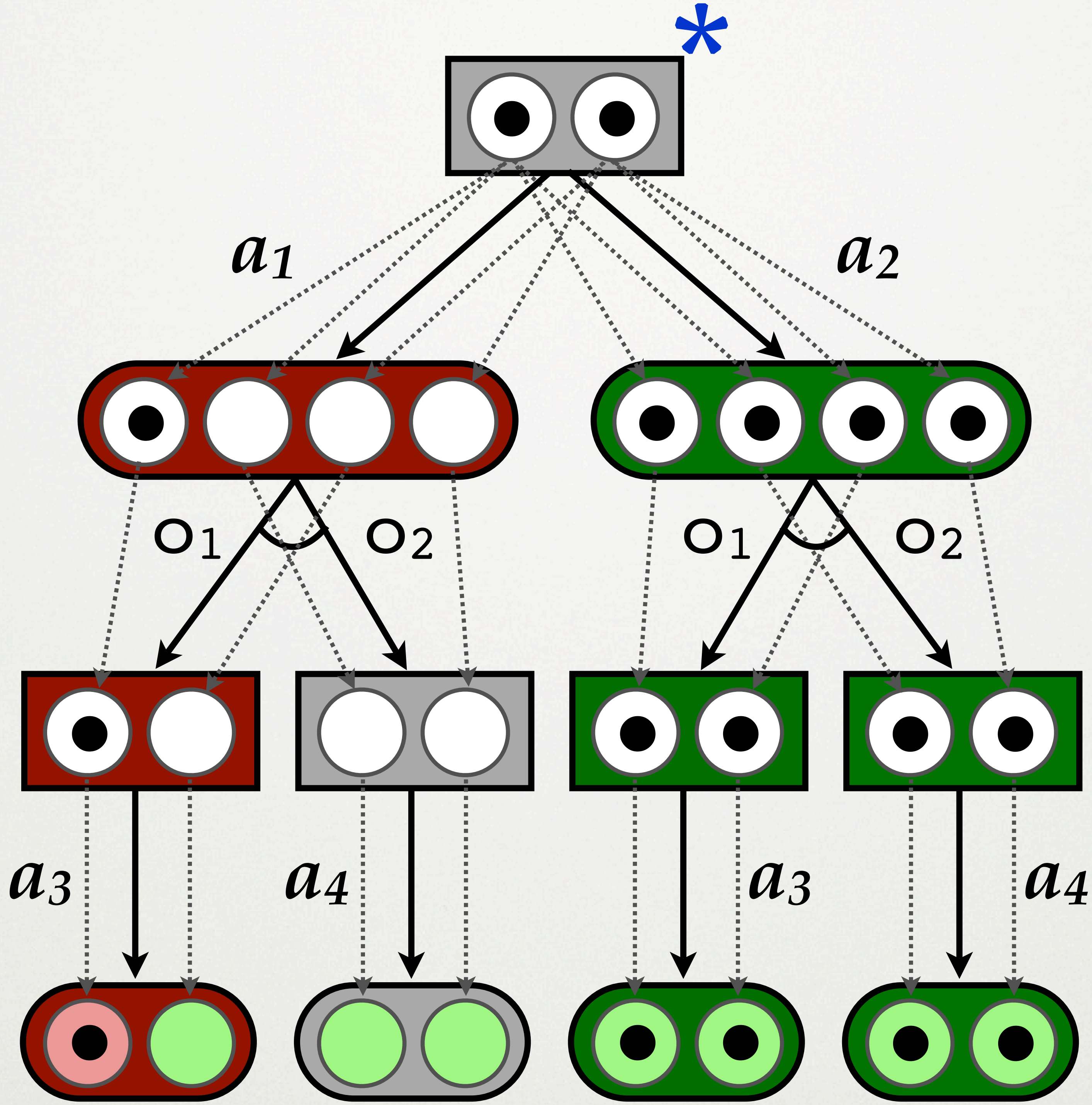
# DBU



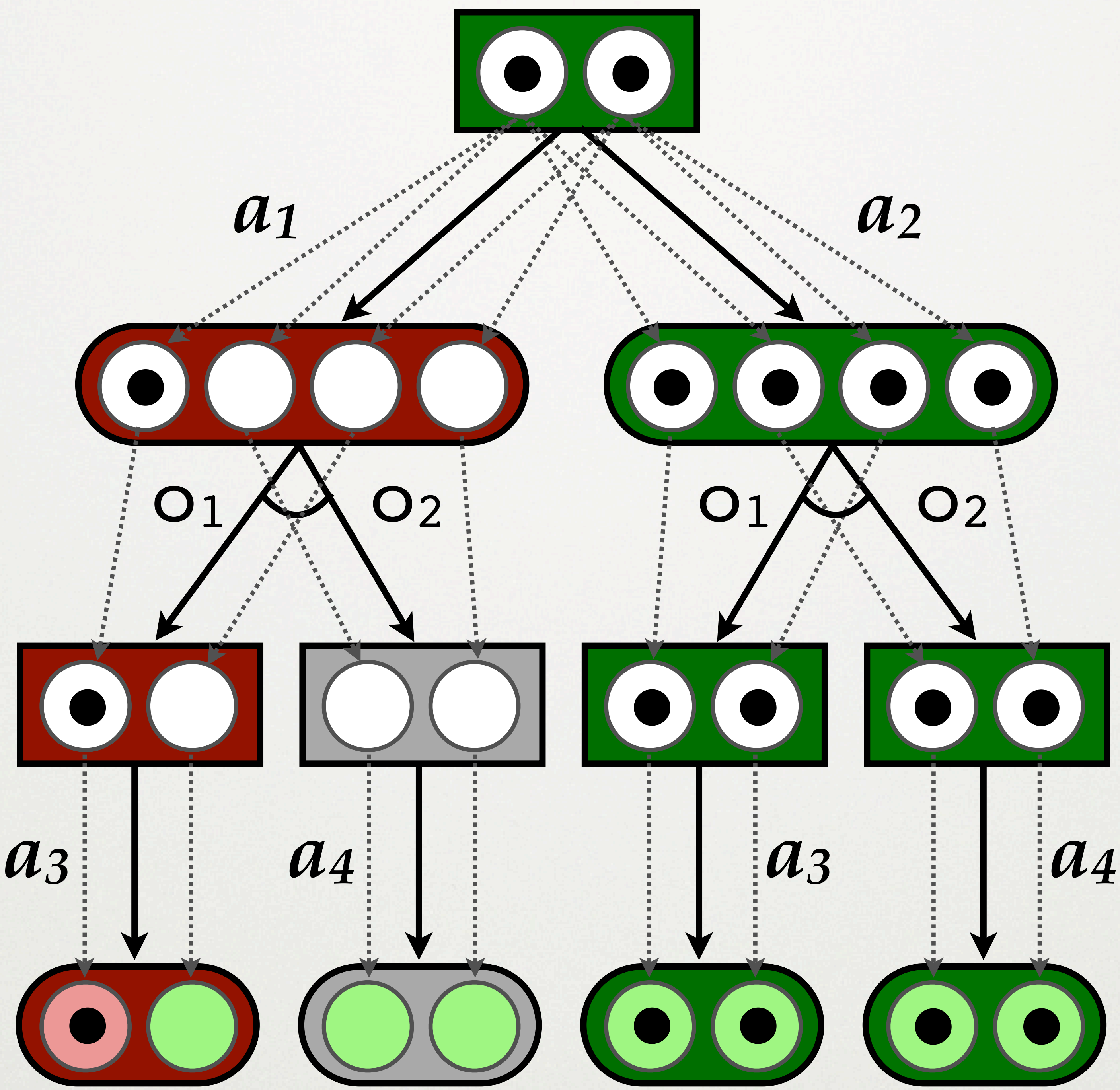
# DBU



# DBU



# DBU



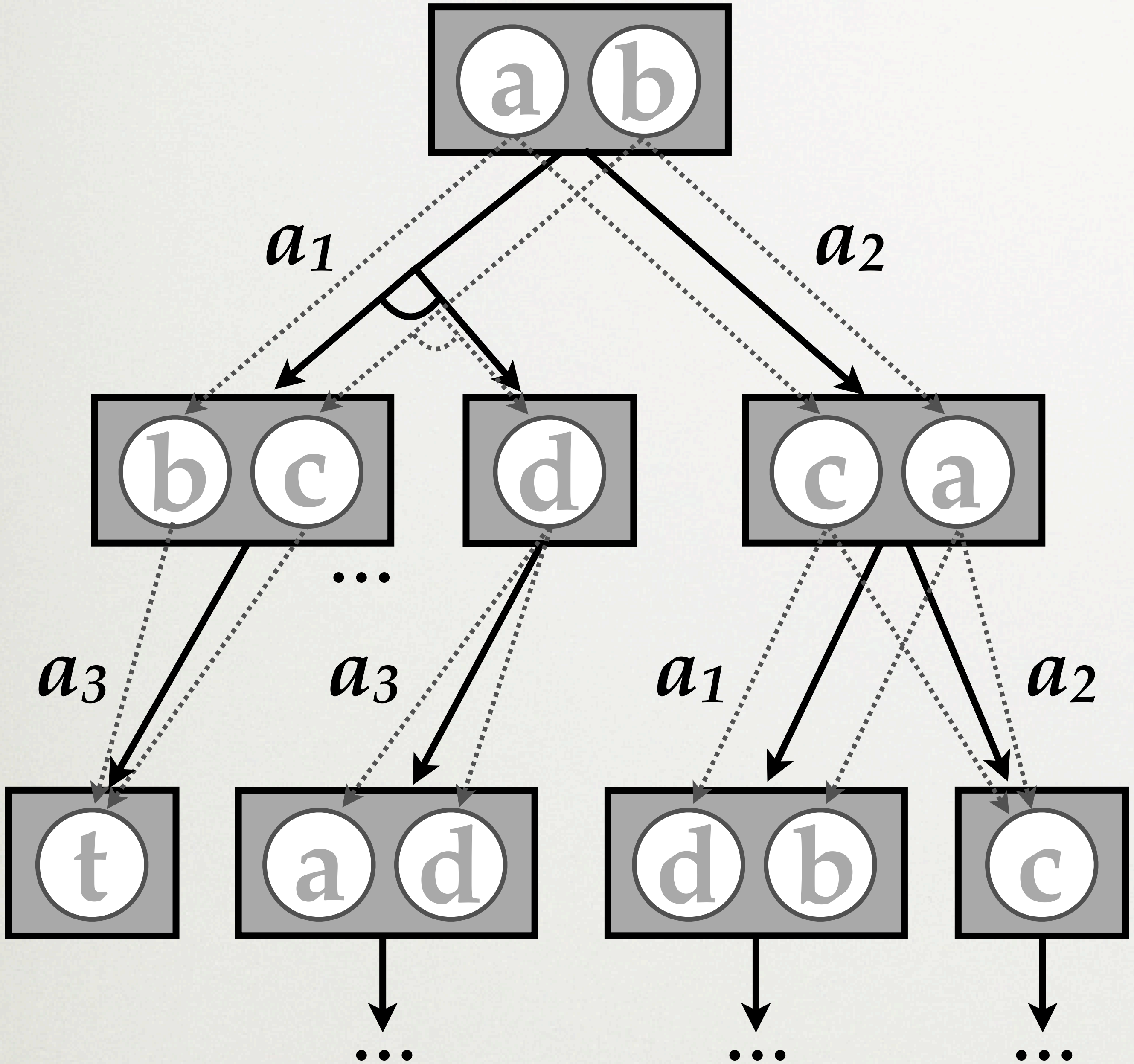
# Belief State AND/OR Graph Search

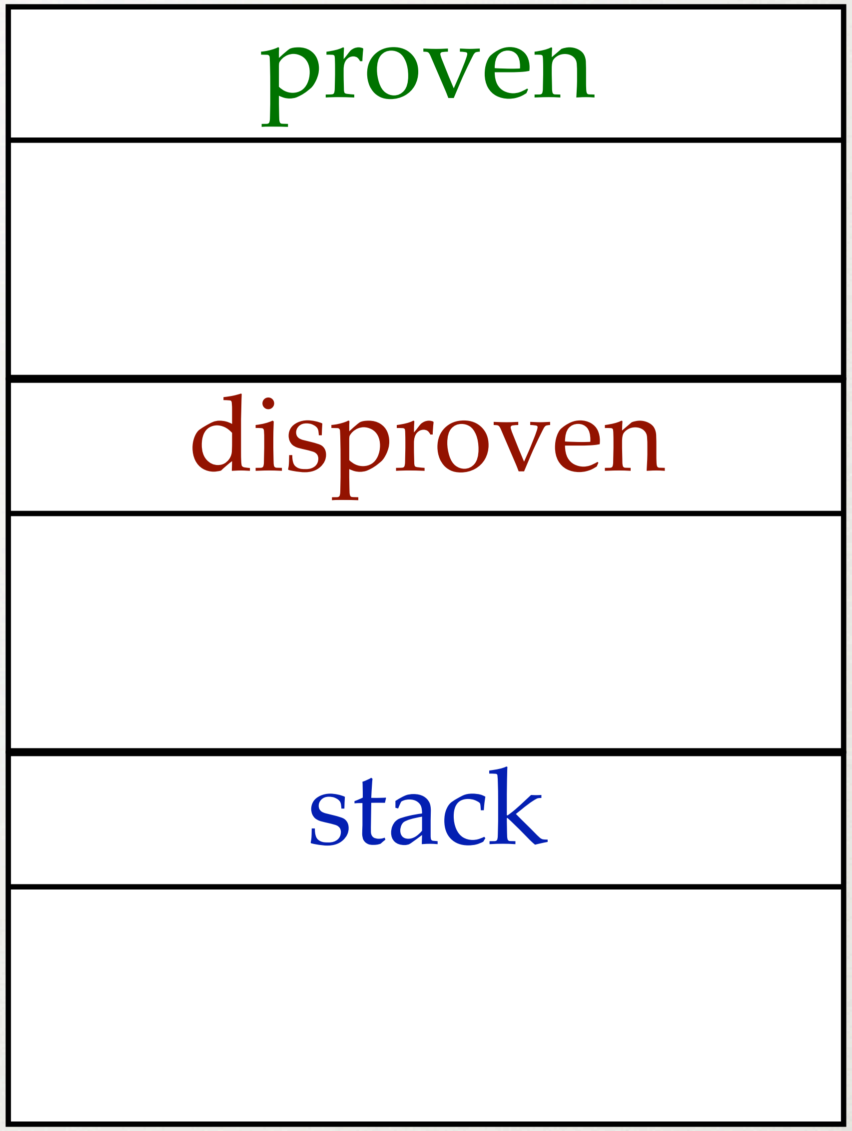
- Identifying repeated nodes can prevent:
  - re-doing previous work
  - wasting effort considering cyclic plans
- Previous work: exactly repeated nodes
  - [Bertoli et al. (2001)] rely on canonicality of BDD-based belief state representations
  - [Sakuta and Iida (2001)] use hashing on sets of explicitly represented physical states

# Key Observations

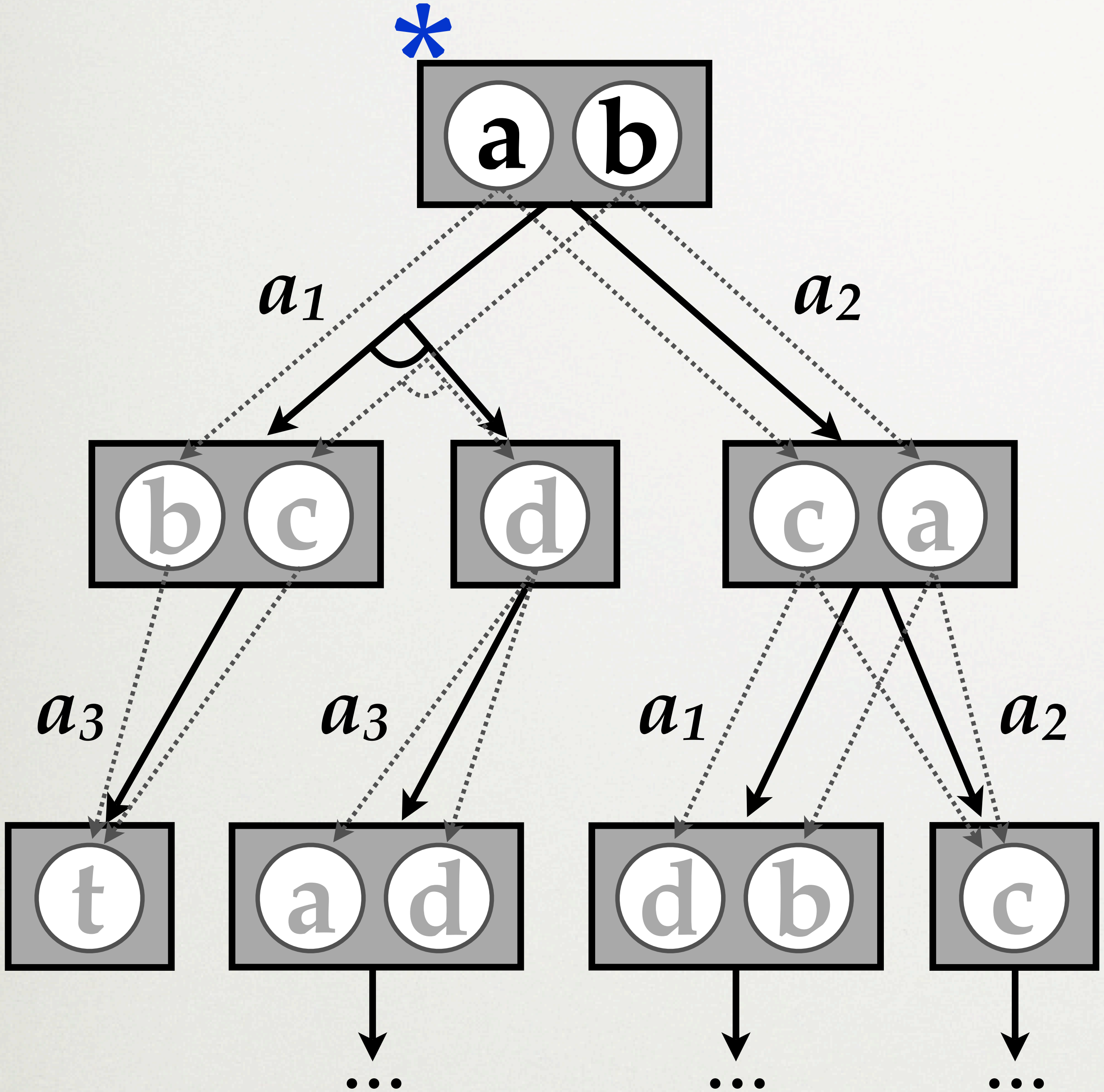
- Since a plan for belief state *b* must work at all of its physical states:
  - a plan for *b* works on all subsets of *b*
  - no plans for *b* ⇒ all supersets of *b* unsolvable
- We will fully exploit these facts by:
  - re-using proofs from supersets
  - re-using disproofs from subsets
  - avoiding generalized cycles (ancestor is subset)

# DFS⊆ Graph Search



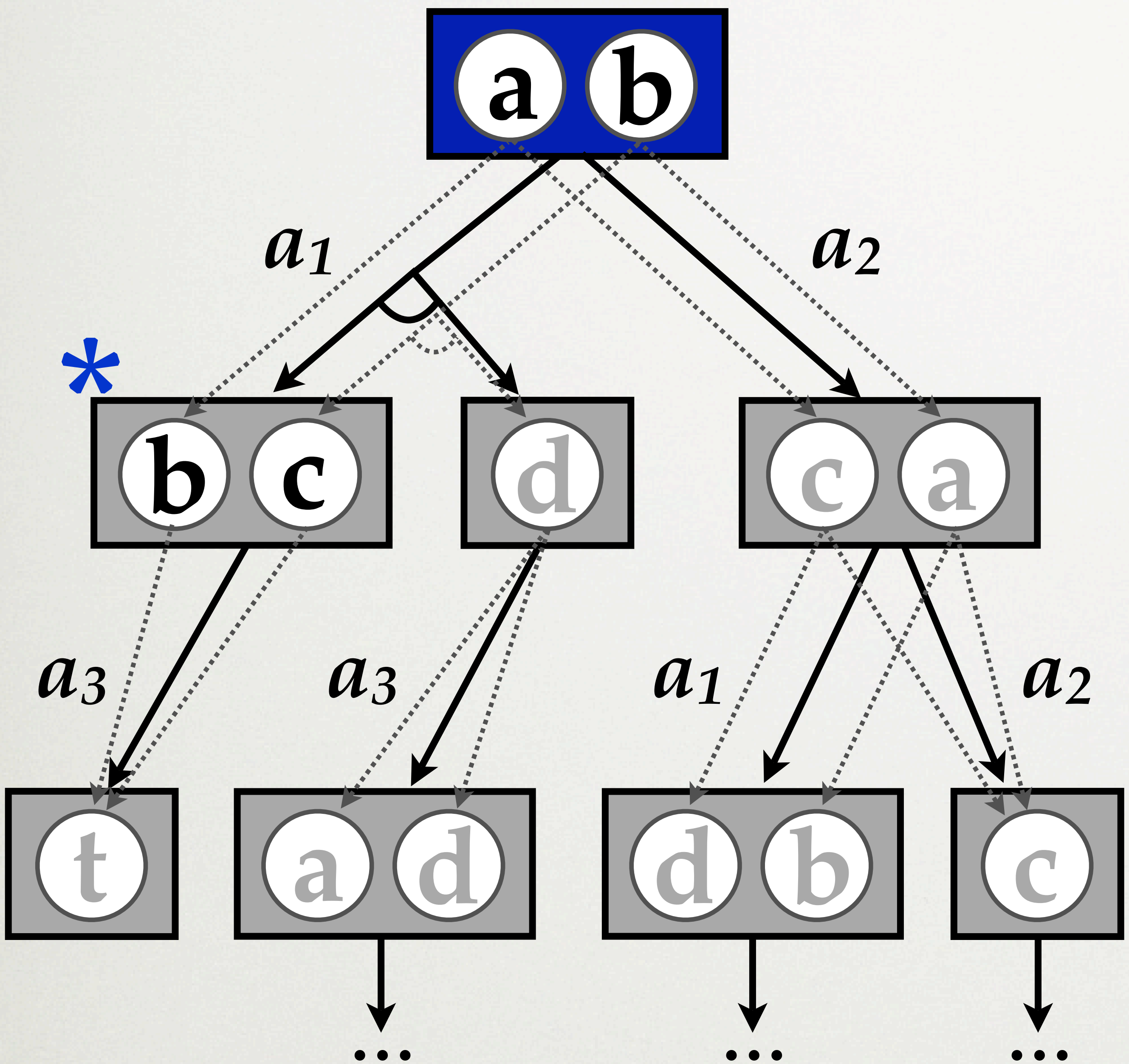



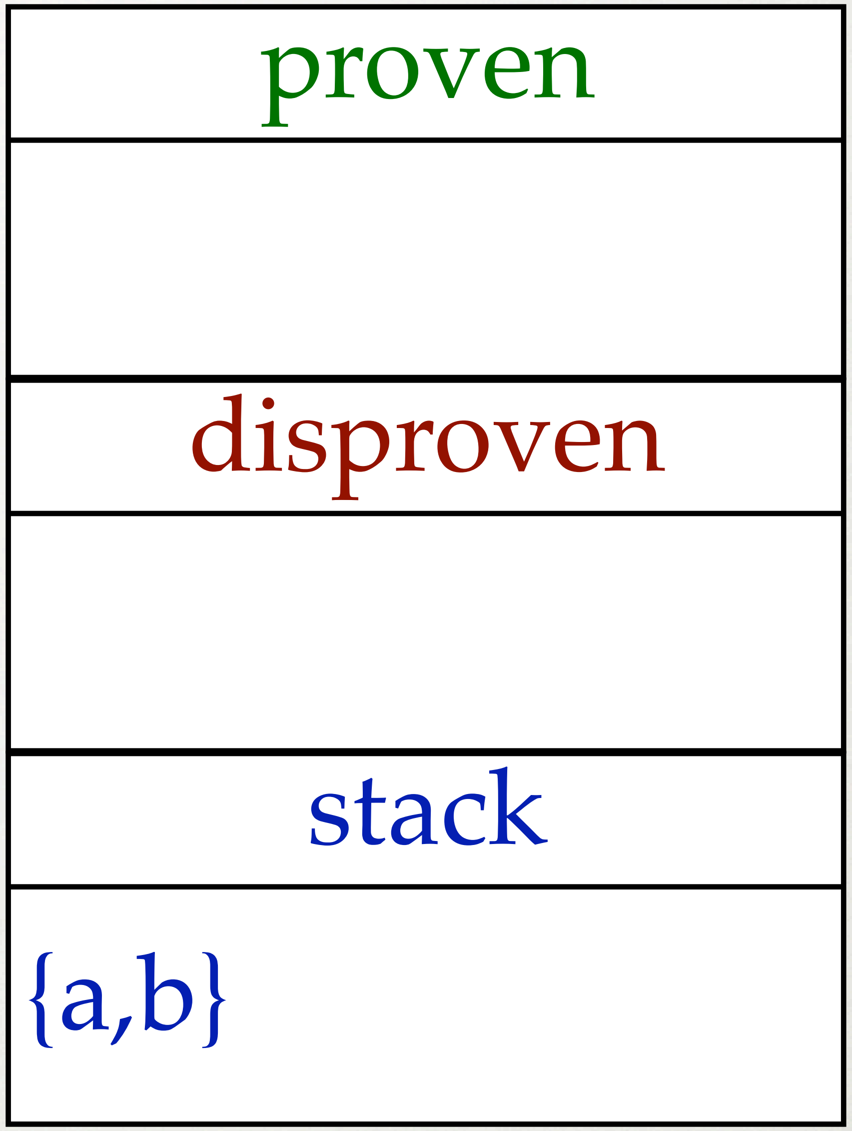
# DFS ⊆ Graph Search



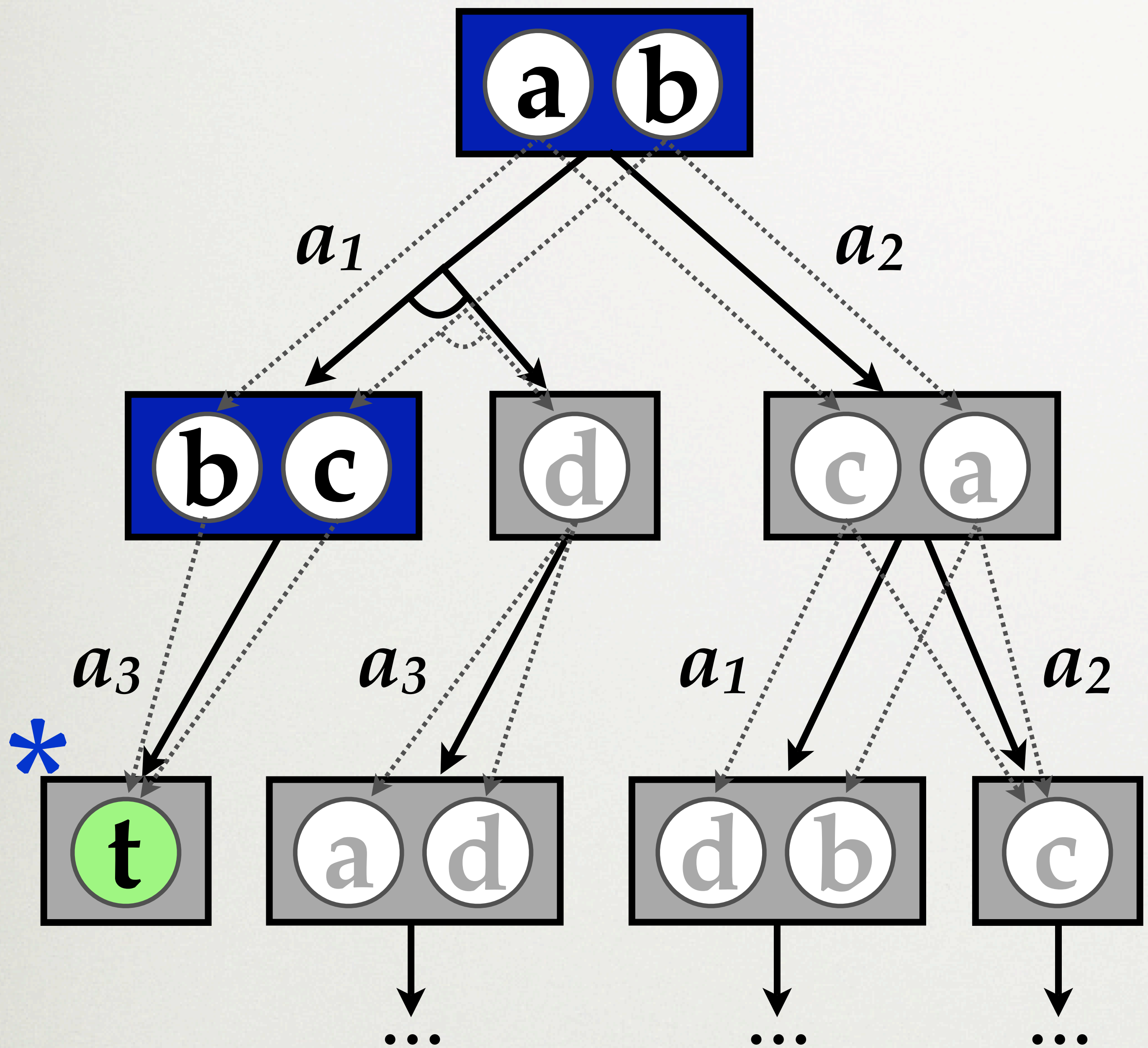

| |
|---|
| proven |
| |
| disproven |
| |
| stack |
| |

# DFS⊆ Graph Search



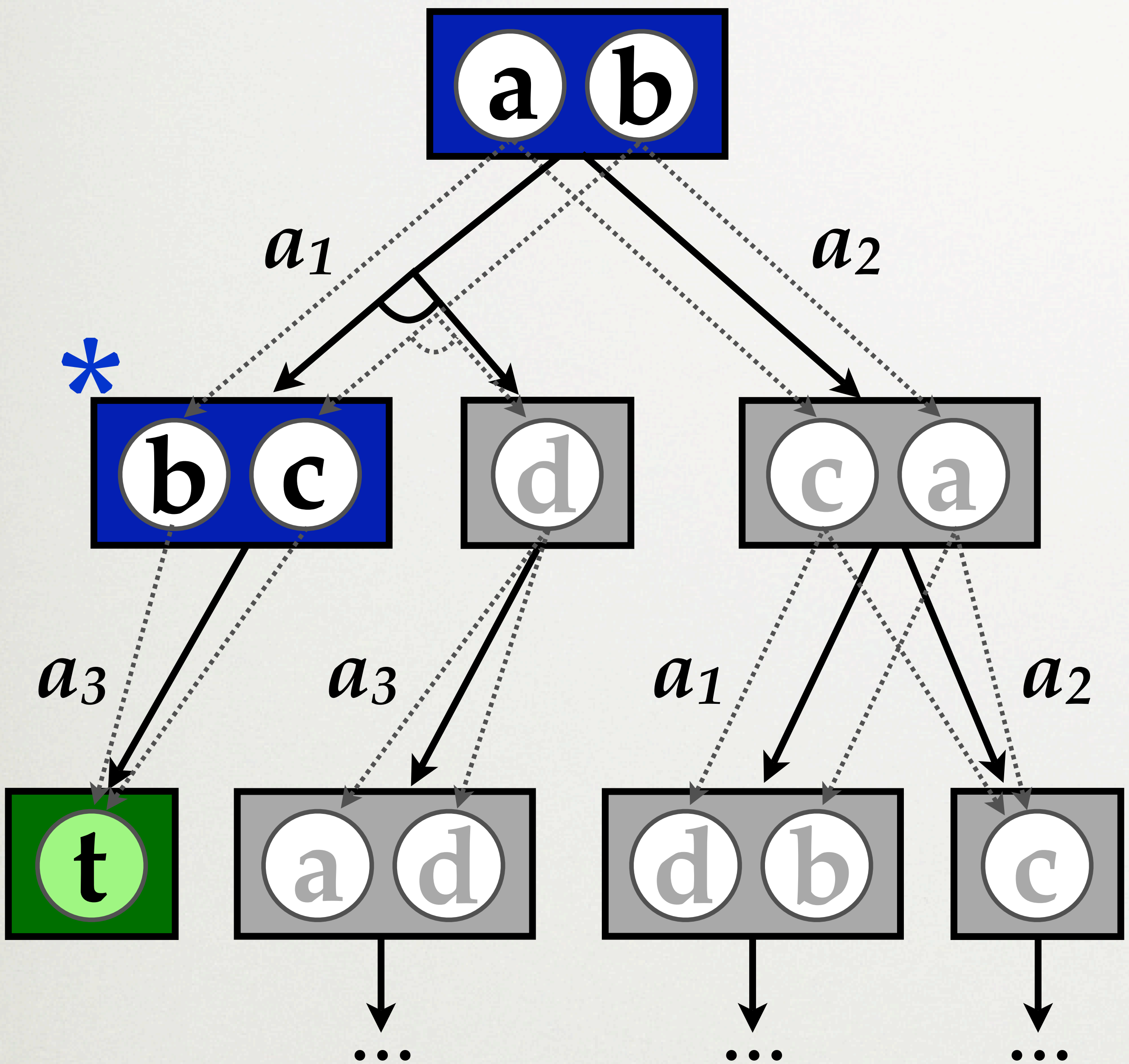



# DFS$\subseteq$ Graph Search



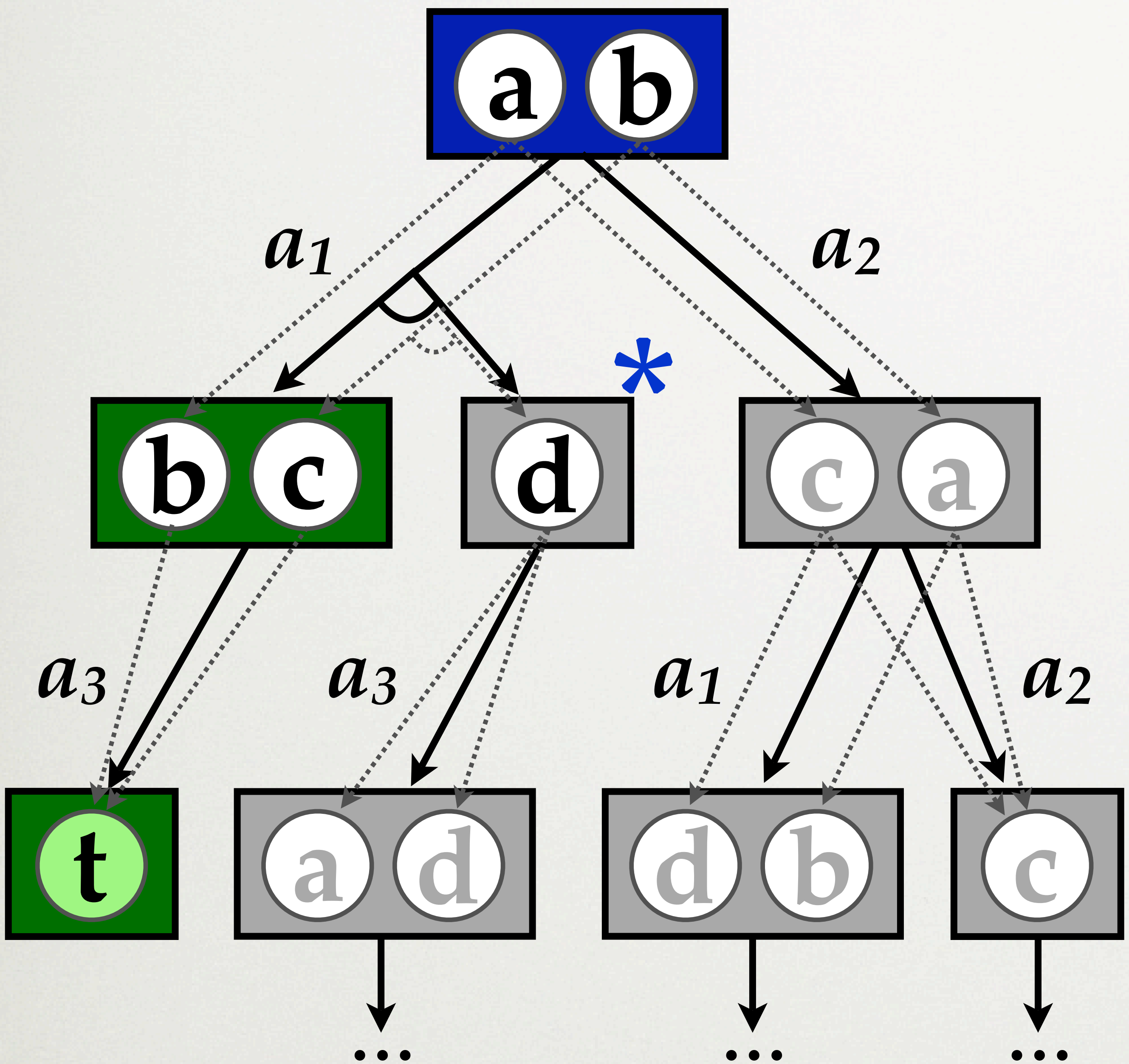

| proven |
| --- |
| |
| **disproven** |
| |
| **stack** |
| {a,b} {b,c} |

# DFS⊆ Graph Search



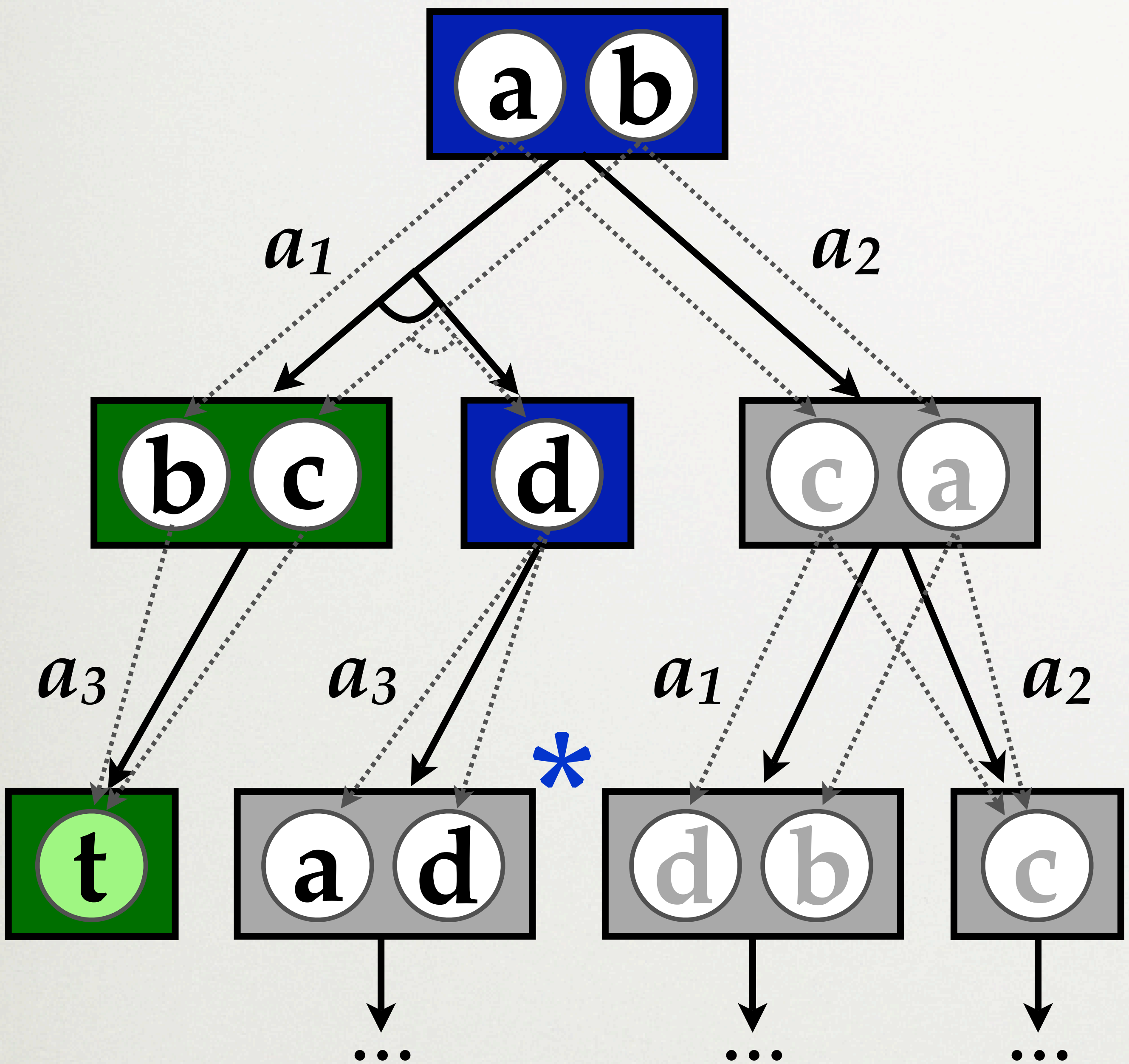

| proven |
|---|
| {t} |
| disproven |
| |
| stack |
| {a,b} {b,c} |

# DFS⊆ Graph Search



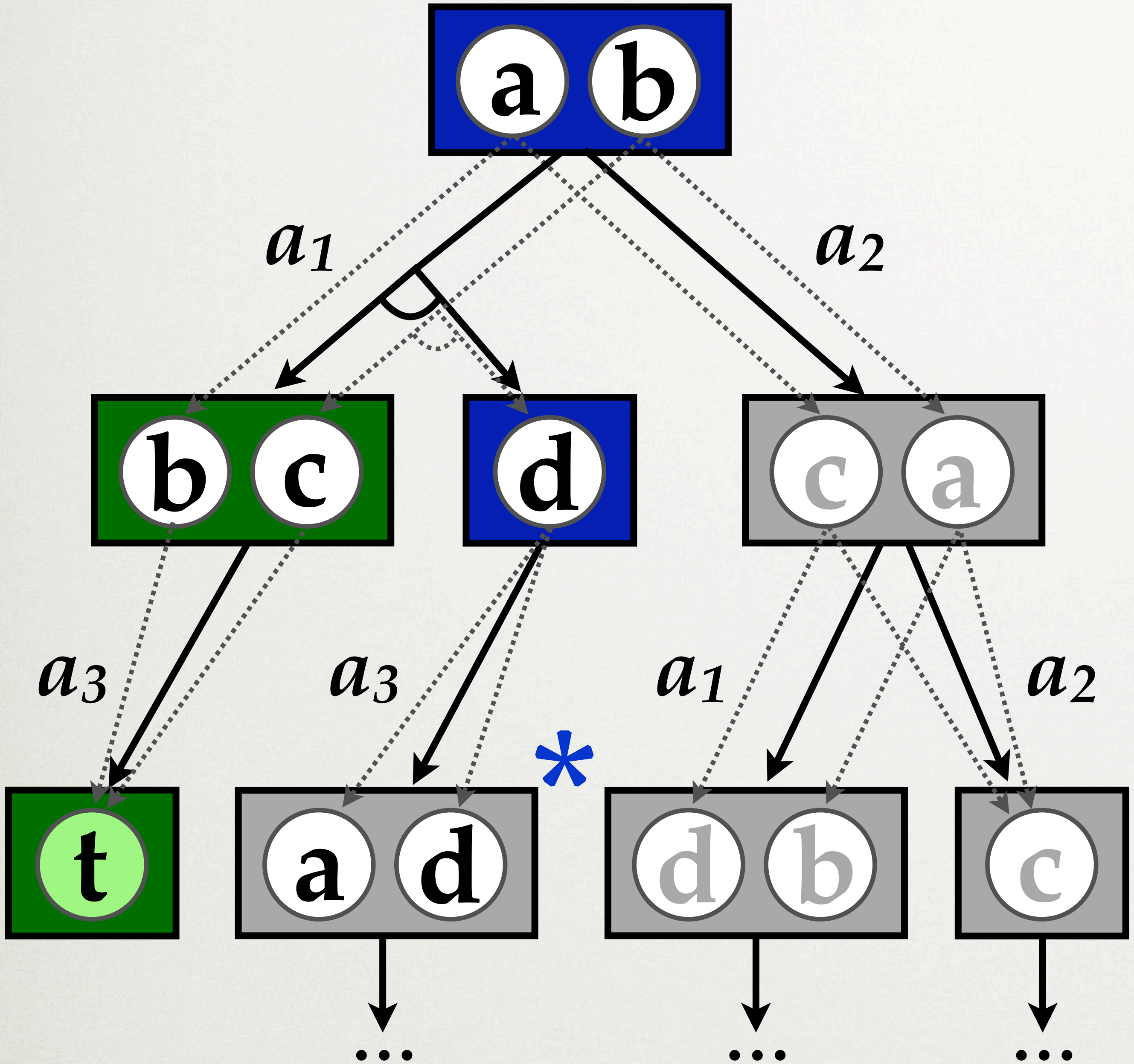

| proven |
|---|
| {t} {b,c} |
| **disproven** |
| |
| **stack** |
| {a,b} |

# DFS⊆ Graph Search



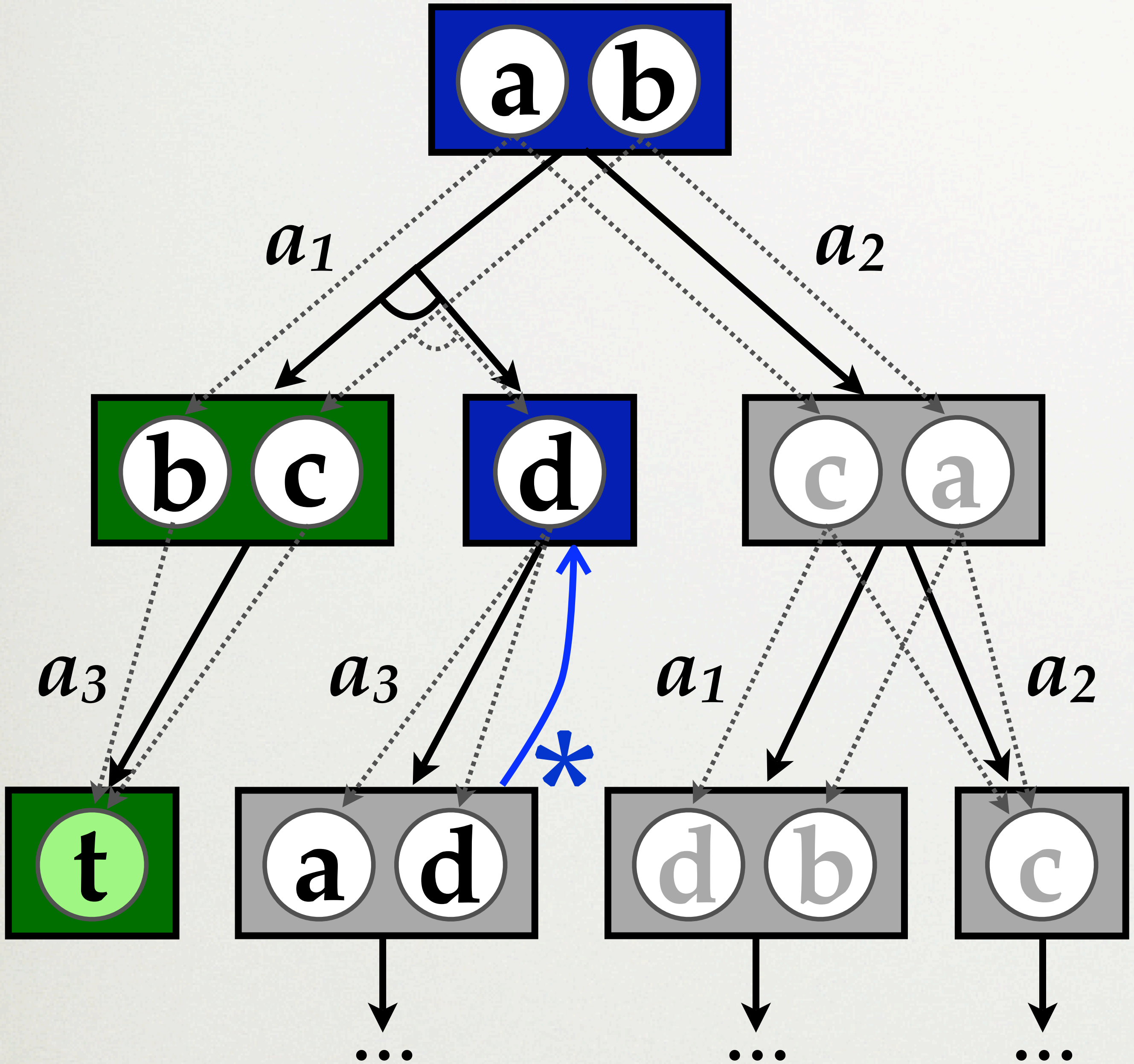

| proven |
| --- |
| {t} {b,c} |
| disproven |
| |
| stack |
| {a,b} {d} |

# DFS⊆ Graph Search



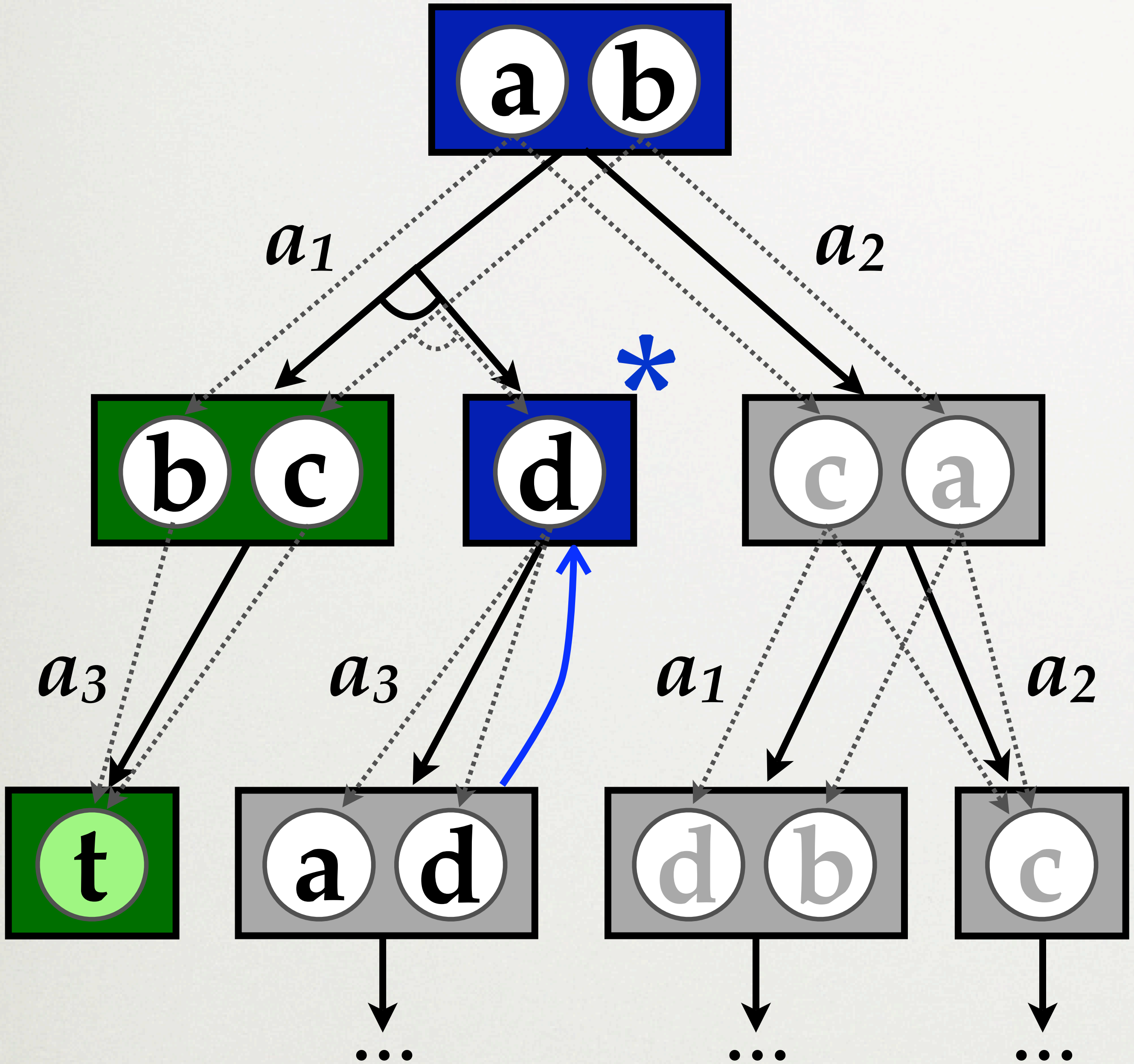

| proven |
|---|
| {t} {b,c} |
| disproven |
| |
| stack |
| {a,b} {d} |

# DFS⊆ Graph Search



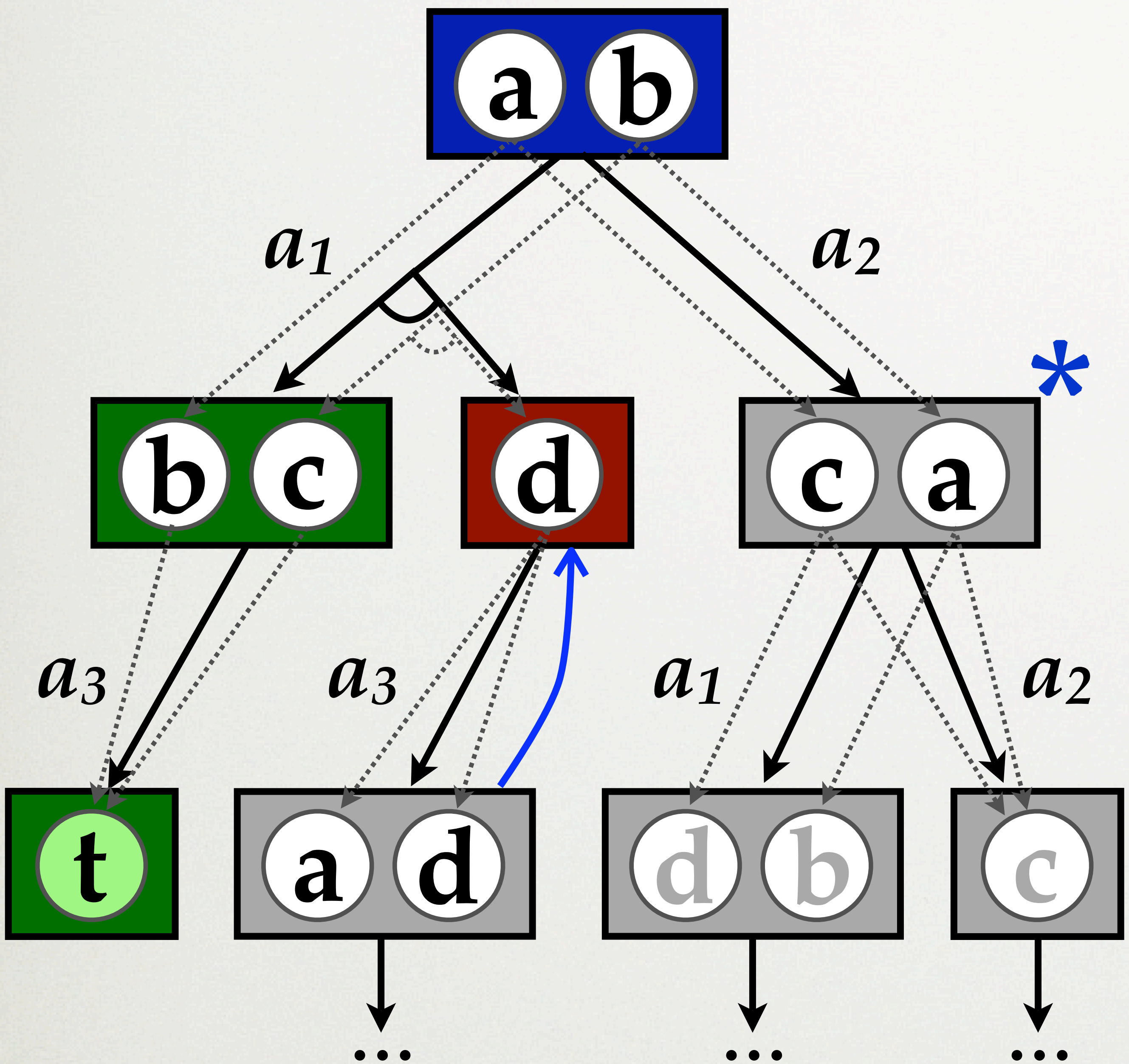

| proven |
|---|
| {t} {b,c} |
| **disproven** |
| |
| **stack** |
| {a,b} {d} |

# DFS⊆ Graph Search



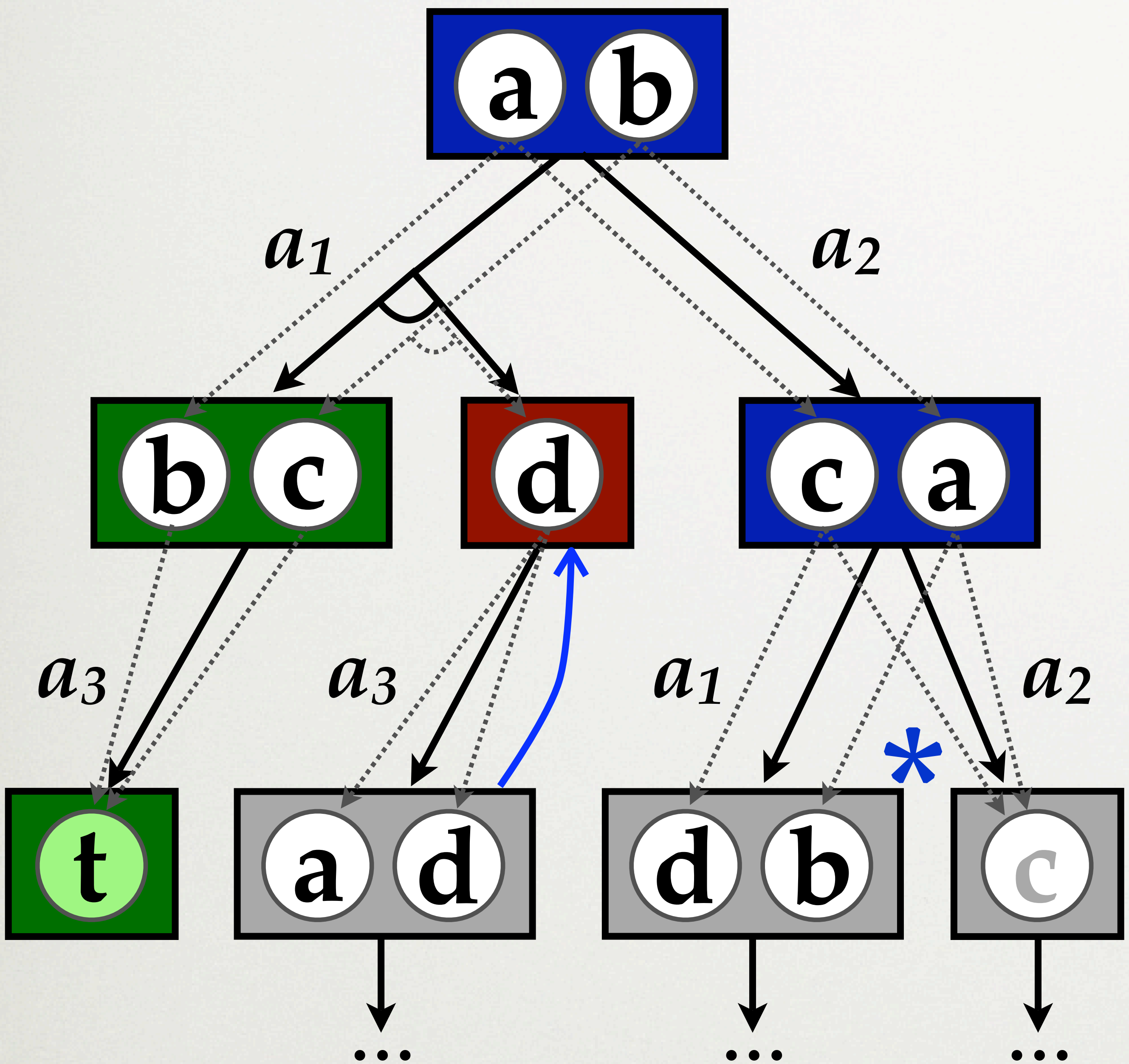

| proven |
|---|
| {t} {b,c} |
| disproven |
| |
| stack |
| {a,b} {d} |

# DFS⊆ Graph Search



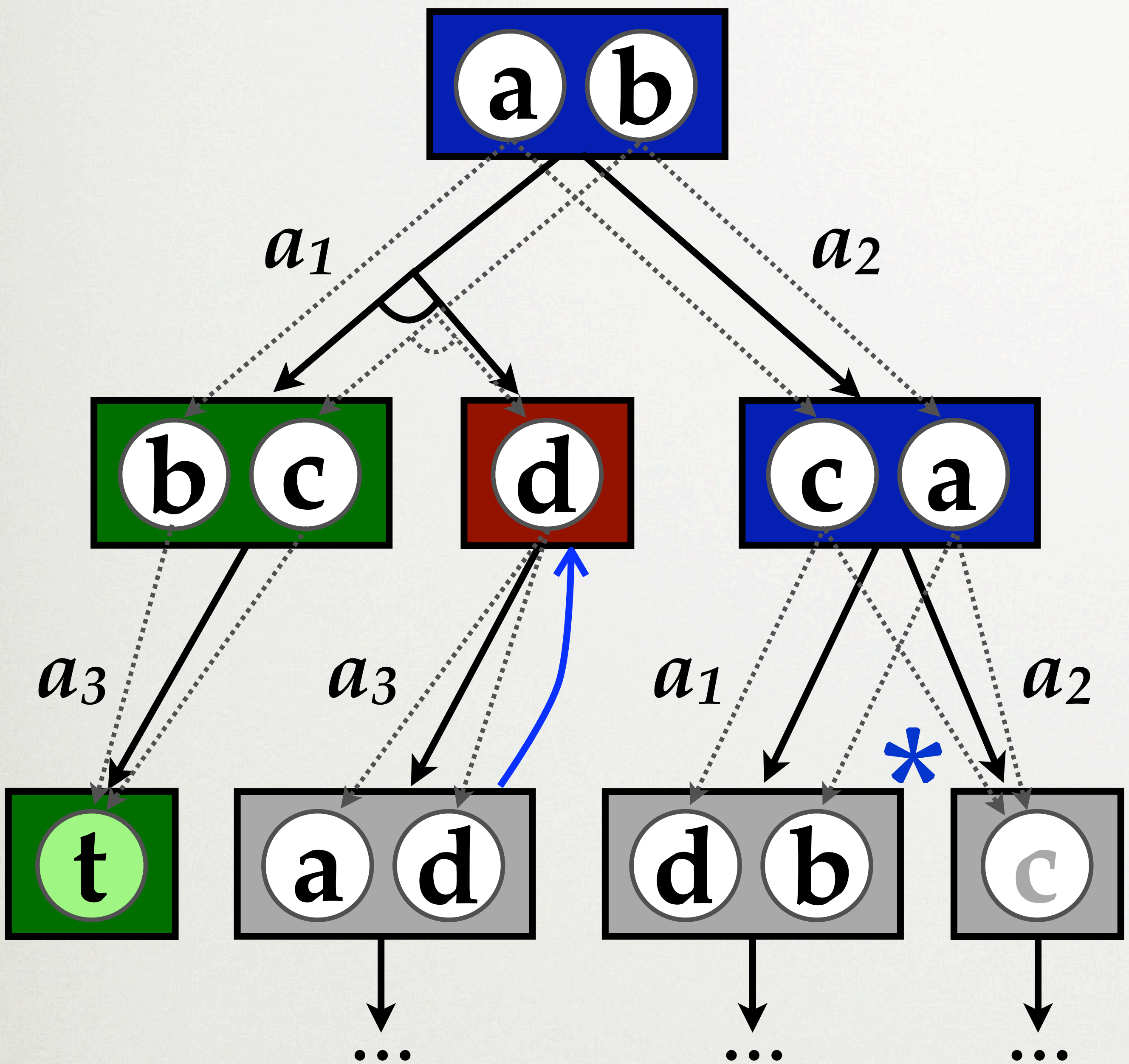

| proven |
|---|
| {t} {b,c} |
| **disproven** |
| {d} |
| **stack** |
| {a,b} |

# DFS⊆ Graph Search



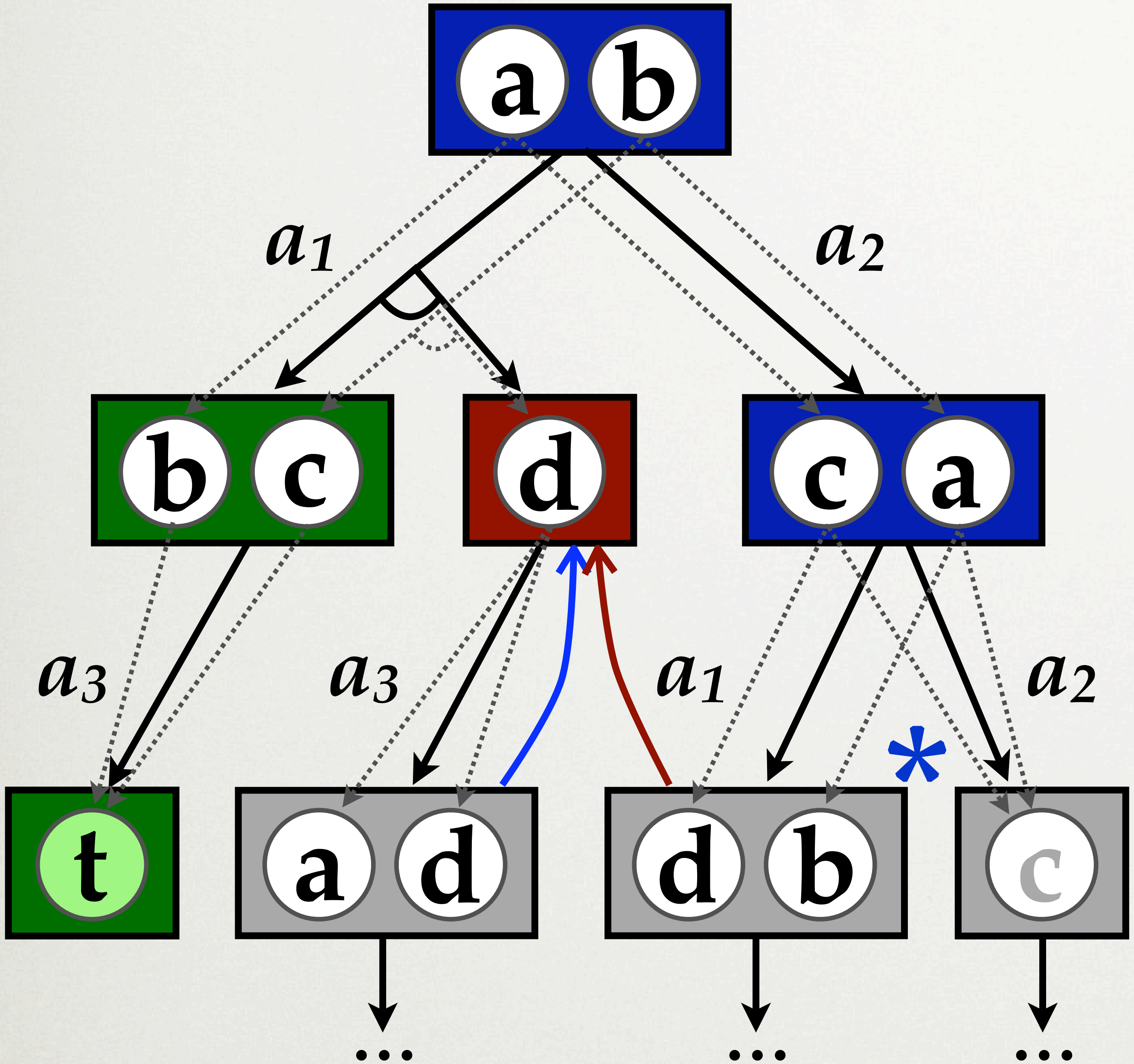

| proven |
|---|
| {t} {b,c} |
| **disproven** |
| {d} |
| **stack** |
| {a,b} {c,a} |

# DFS⊆ Graph Search



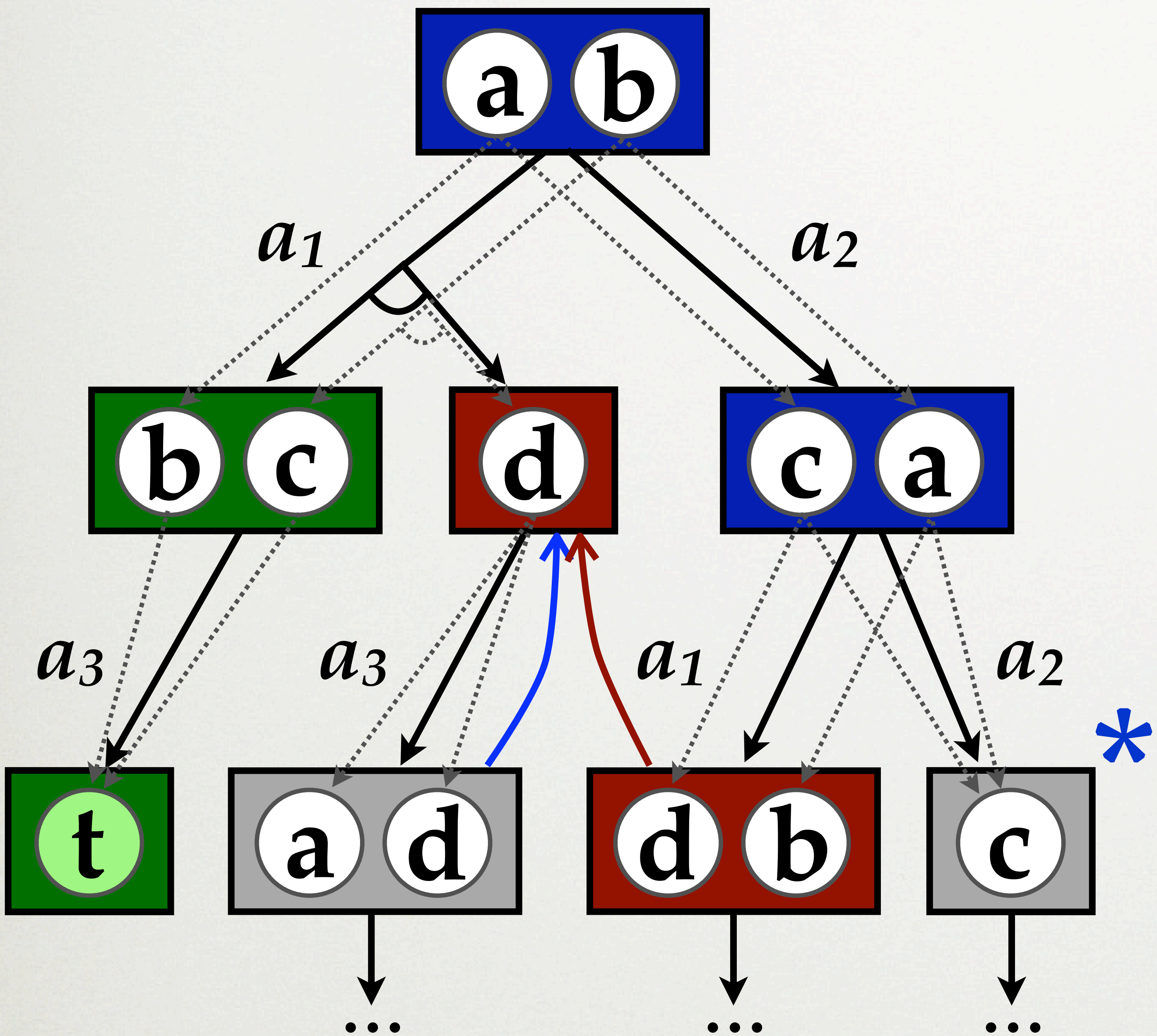

| proven |
| --- |
| {t} {b,c} |
| **disproven** |
| {d} |
| **stack** |
| {a,b} {c,a} |

# DFS⊆ Graph Search



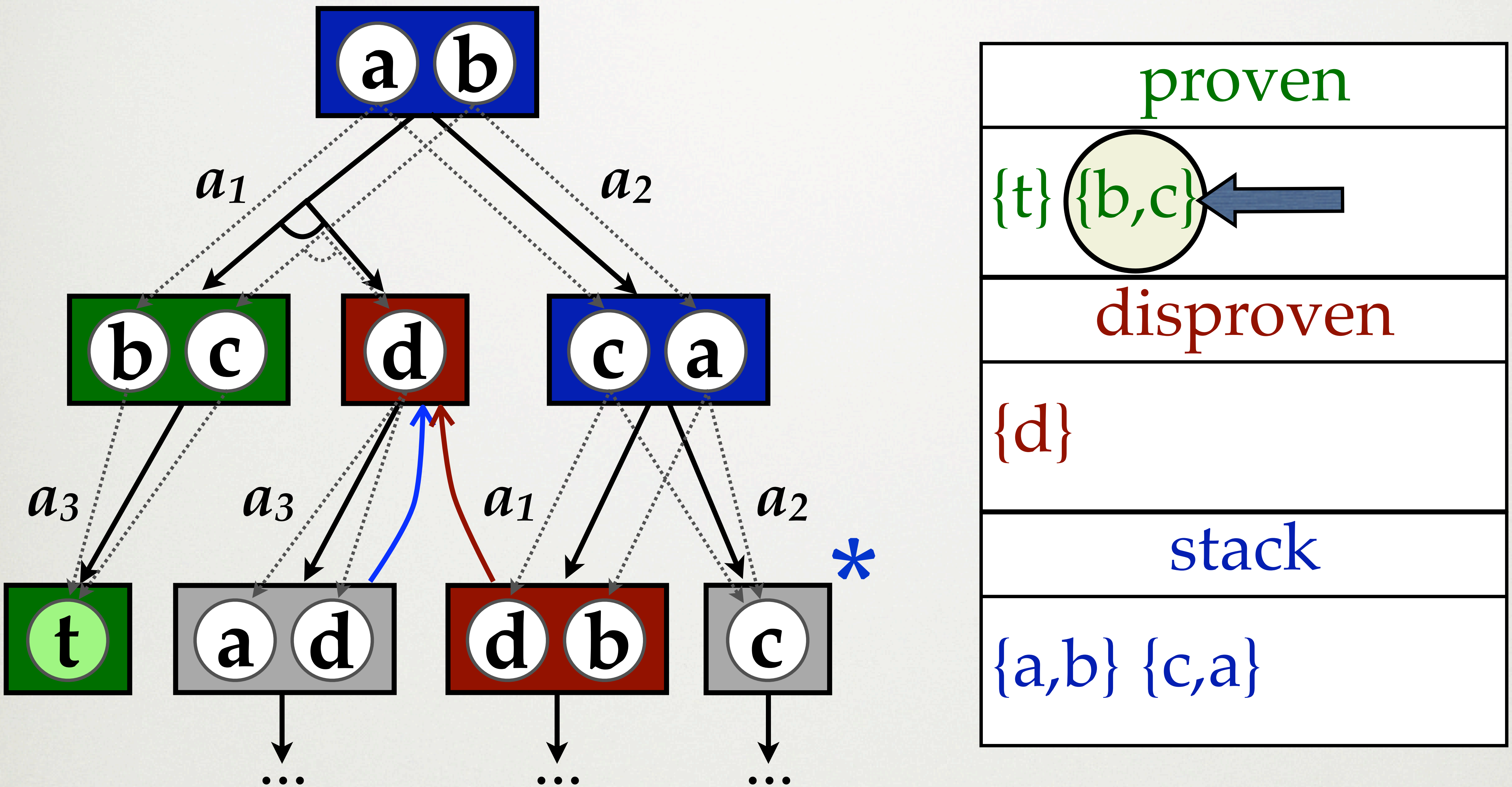

| proven |
| --- |
| {t} {b,c} |
| disproven |
| {d} |
| stack |
| {a,b} {c,a} |

# DFS⊆ Graph Search

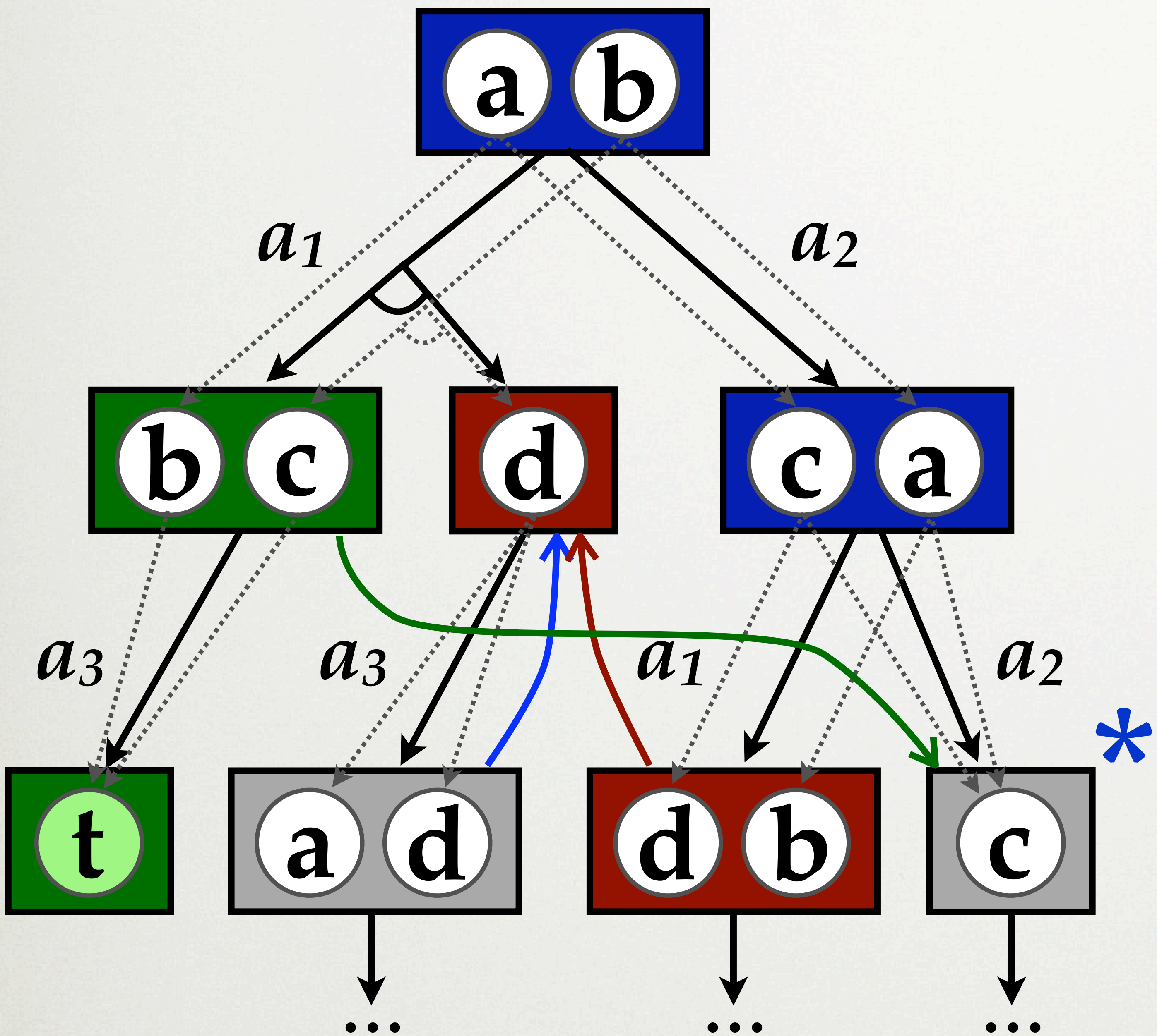


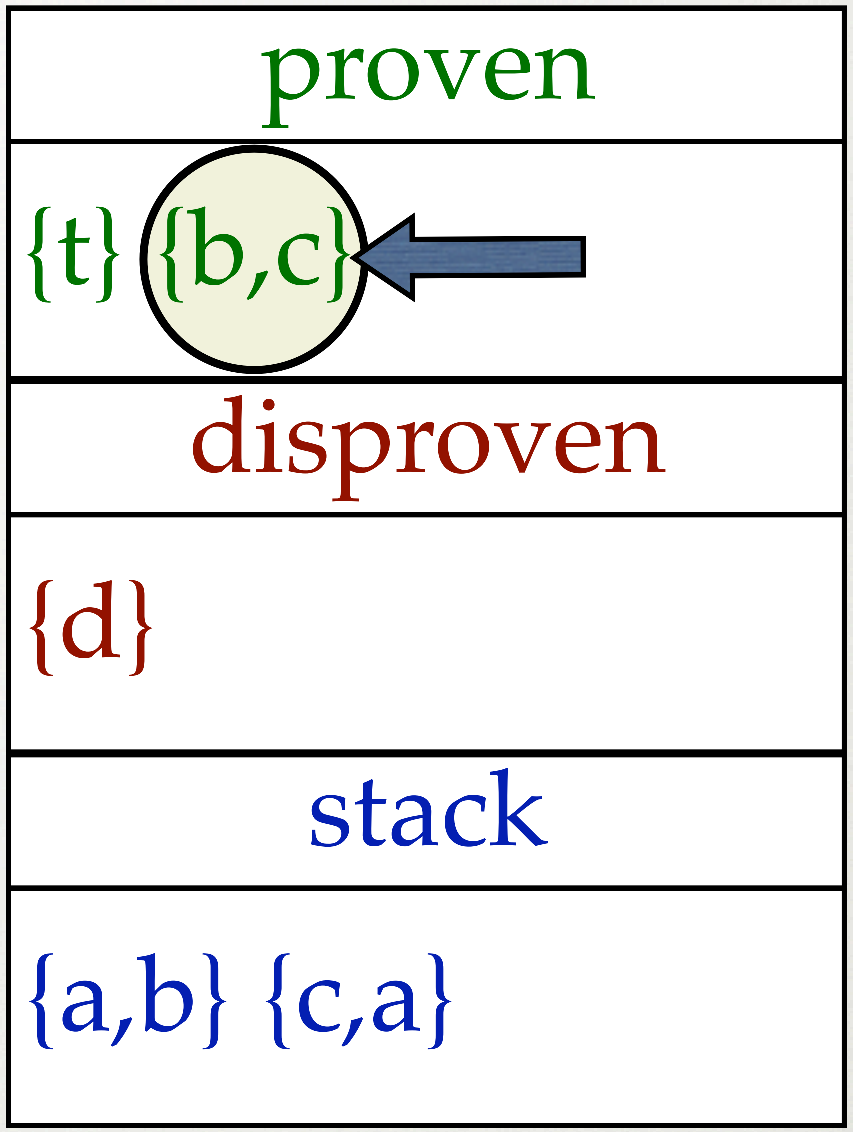

| proven |
|---|
| {t} {b,c} |
| **disproven** |
| {d} |
| **stack** |
| {a,b} {c,a} |

# DFS$\subseteq$ Graph Search

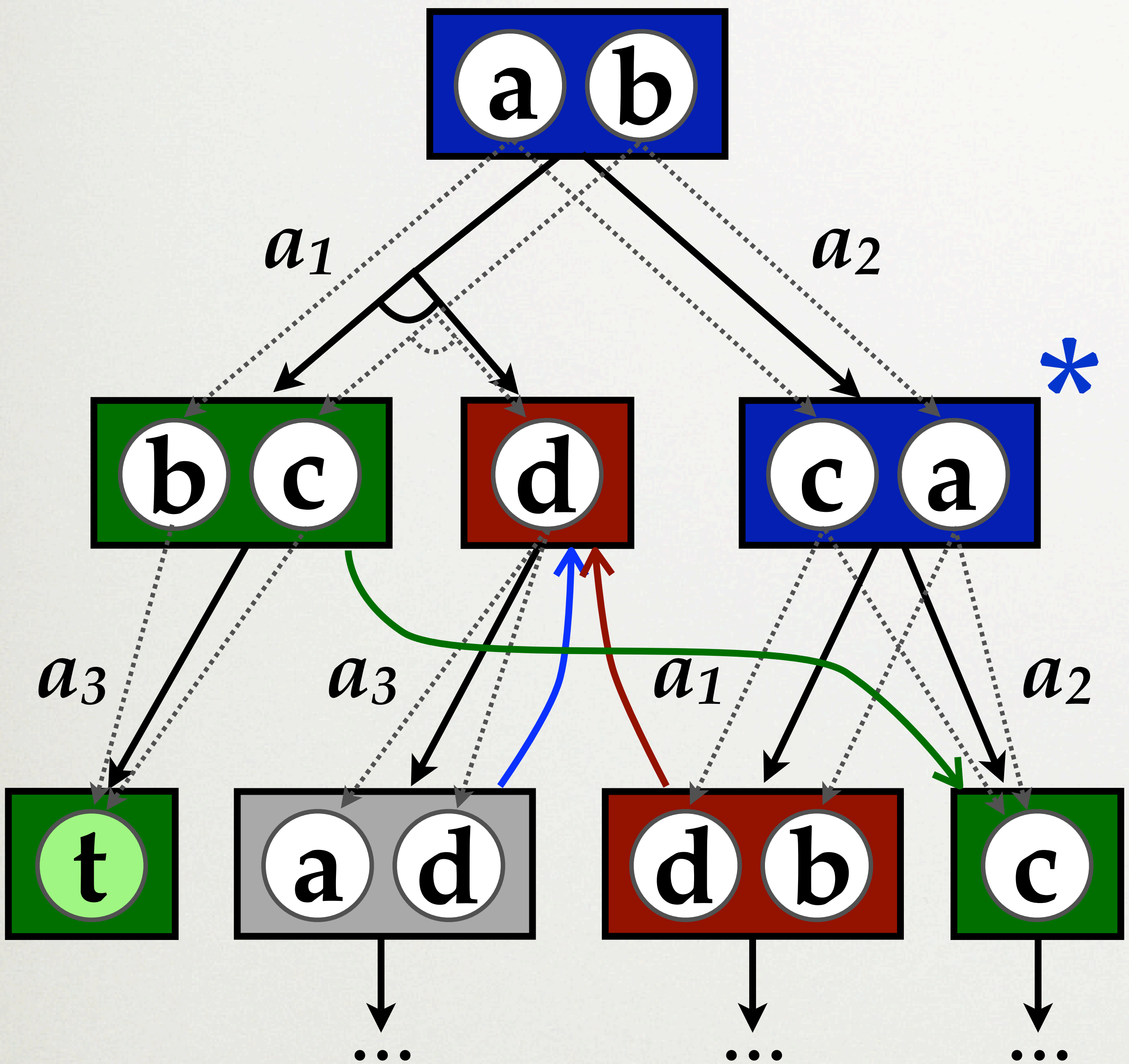


# DFS⊆ Graph Search



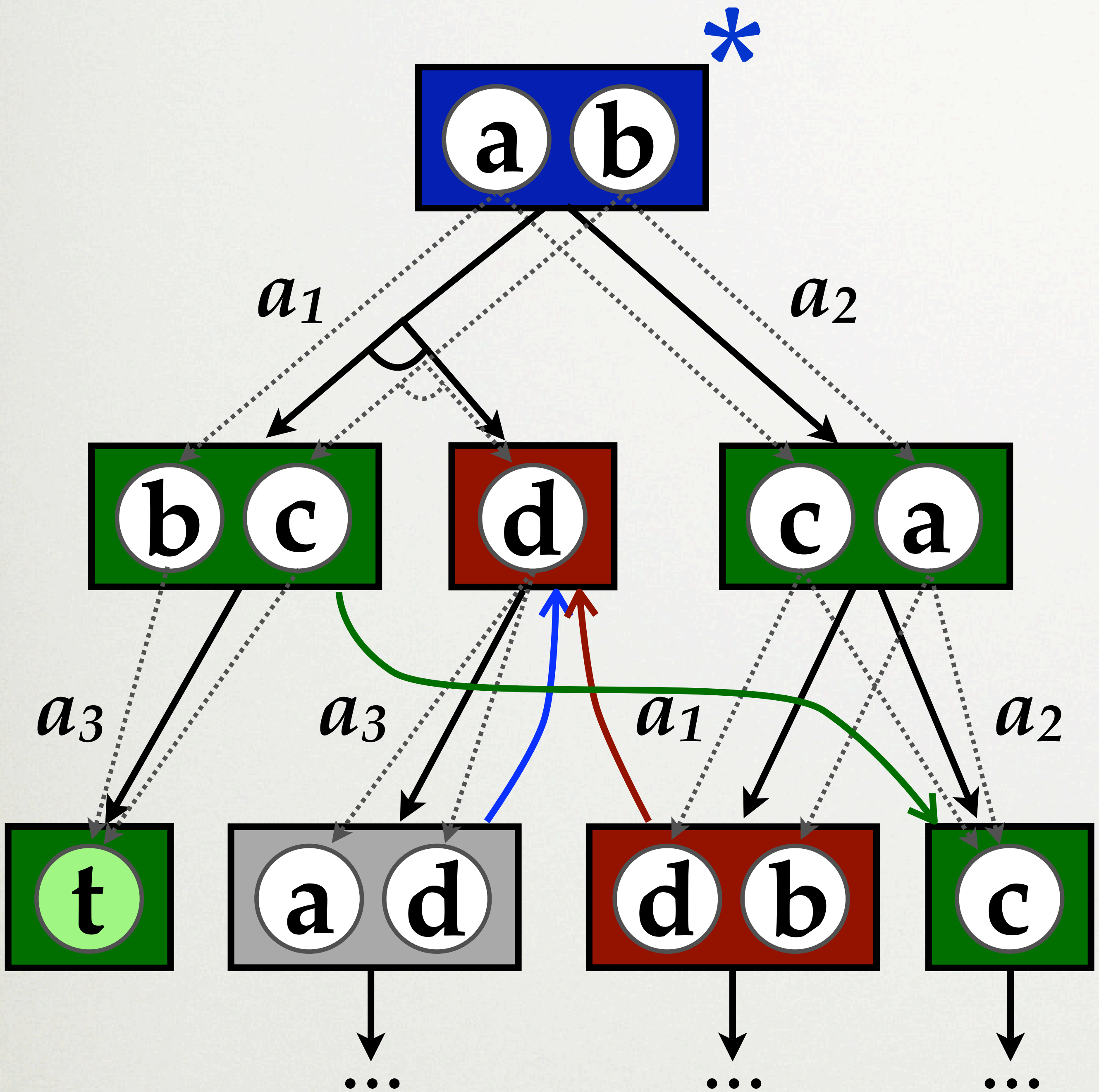



# DFS⊆ Graph Search



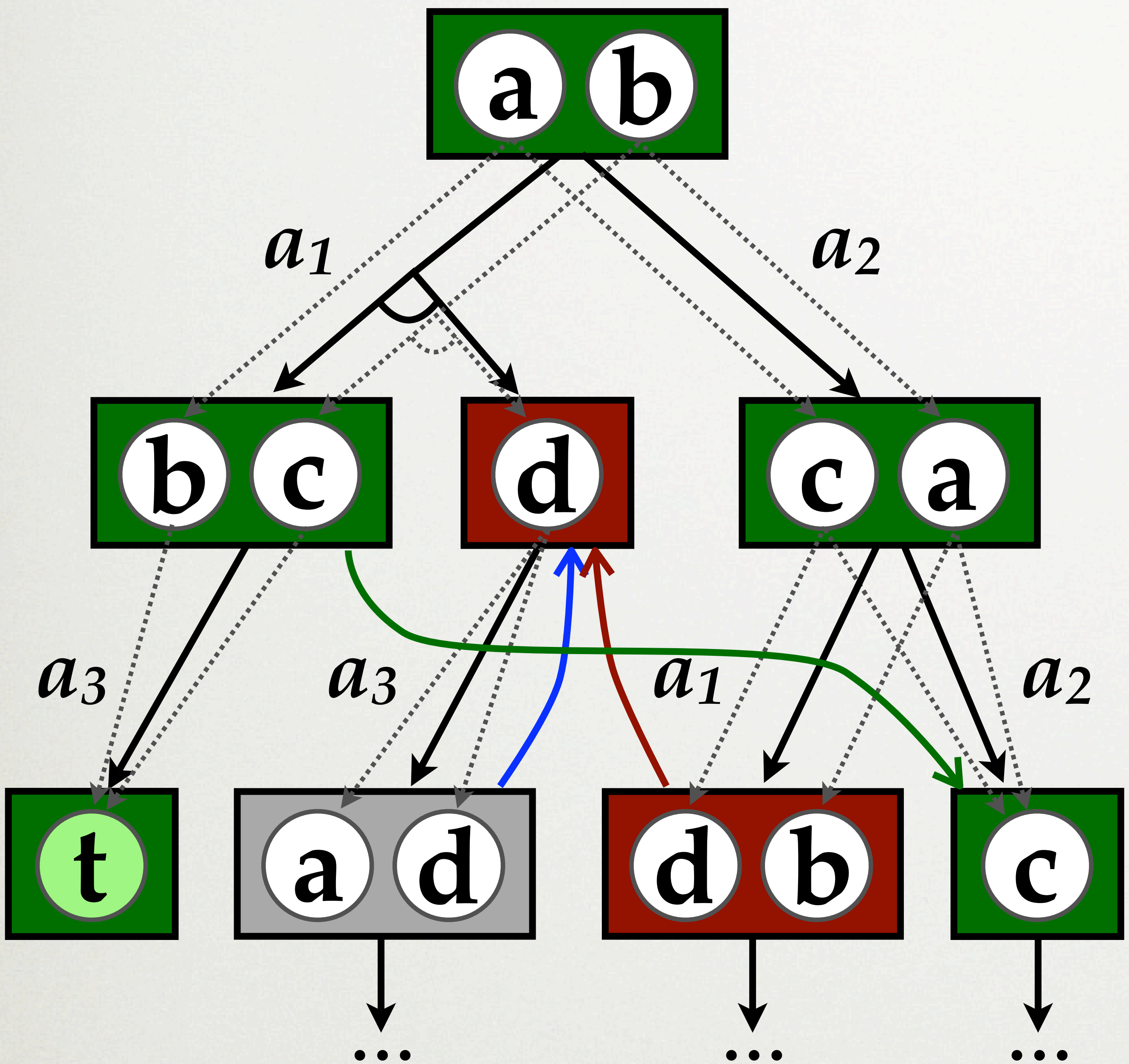

| proven |
| --- |
| {t} {b,c} |
| disproven |
| {d} |
| stack |
| {a,b} {c,a} |

# DFS $\subseteq$ Graph Search




| proven |
|---|
| {t} {b,c} {c,a} |
| **disproven** |
| {d} |
| **stack** |
| {a,b} |

# DFS $\subseteq$ Graph Search




| proven |
| --- |
| {t} {b,c} {c,a} {a,b} |
| **disproven** |
| {d} |
| **stack** |
| {a,b} {c,a} |

# DFS$\subseteq$ Graph Search

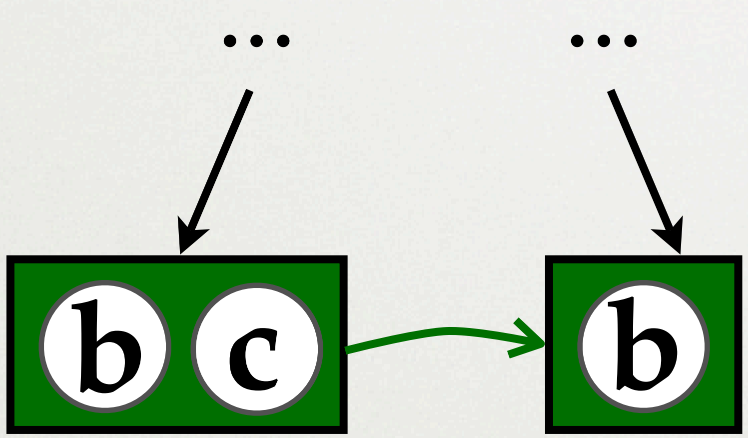
- Previous versions exist for equality case
- Idea: memoization + cycle avoidance
  - mark belief states as proven, disproven, stack
  - at new belief state, find relevant related ones
  - must avoid the Graph History Interaction (GHI) problem

# DBU⊆ Graph Search

- No graph version previously existing
- Modifications are similar to **DFS**, except subsumption relationships can change

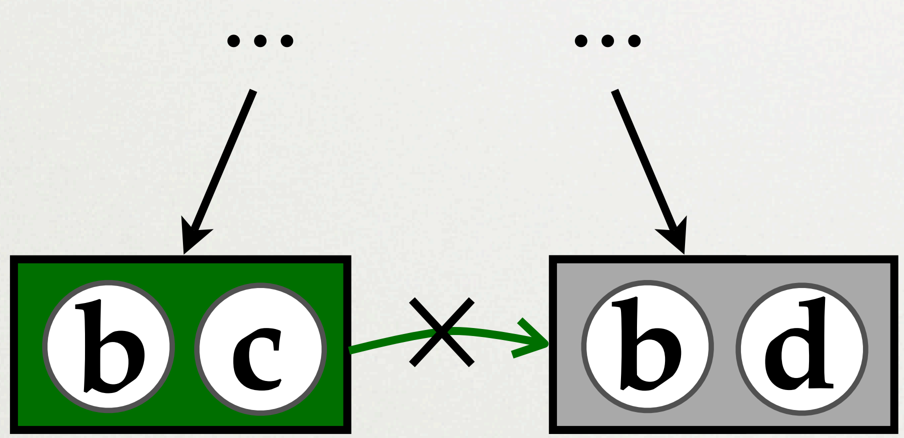
# DBU⊆ Graph Search

- No graph version previously existing
- Modifications are similar to **DFS**, except subsumption relationships can change



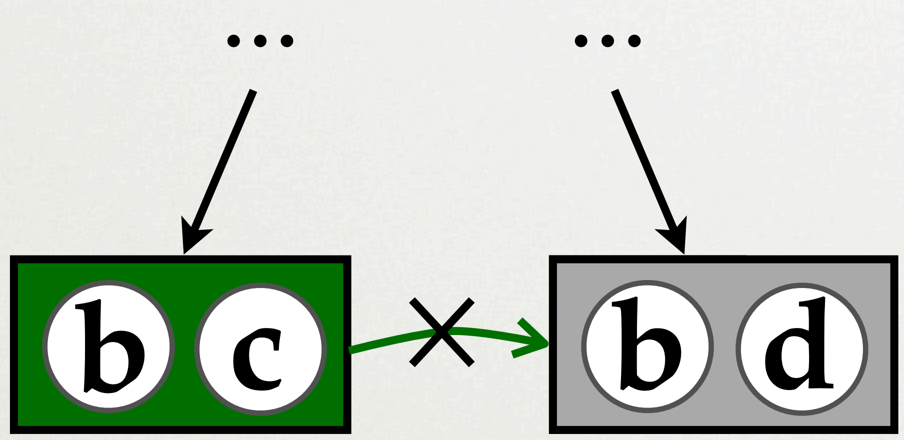
# DBU⊆ Graph Search

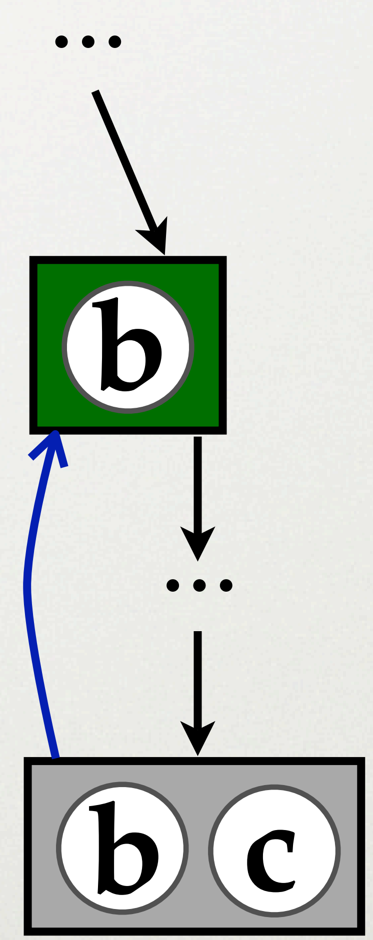
- No graph version previously existing
- Modifications are similar to **DFS**, except subsumption relationships can change



# DBU⊆ Graph Search

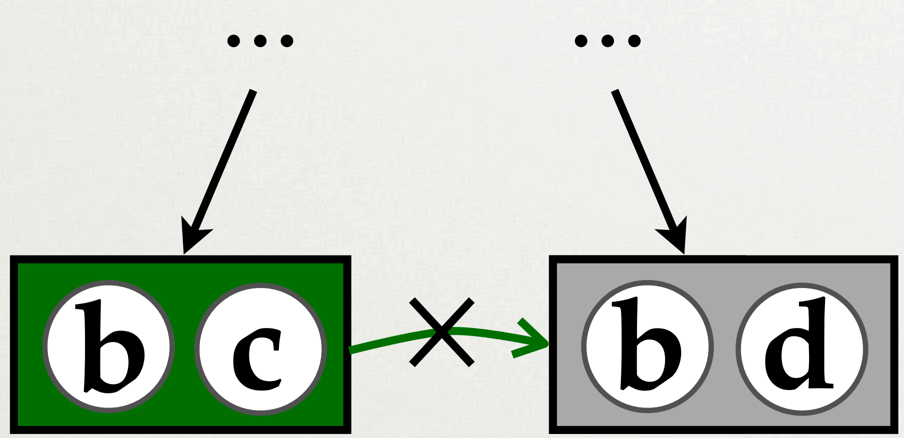
- No graph version previously existing
- Modifications are similar to **DFS**, except subsumption relationships can change

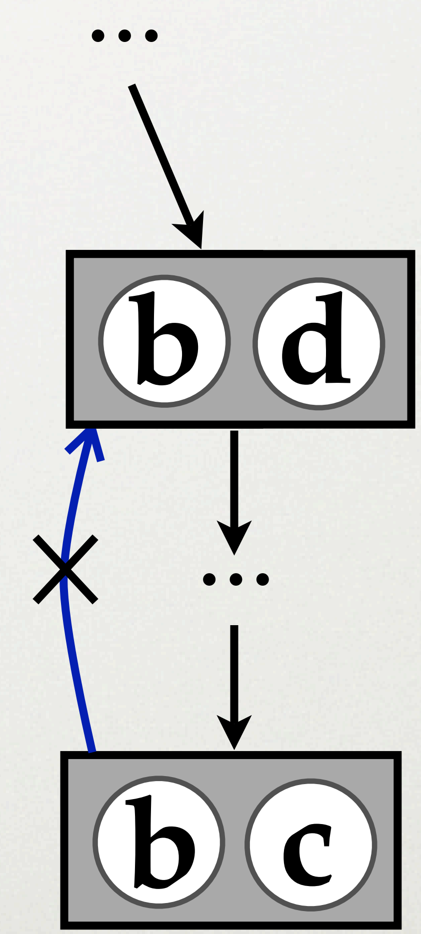


# DBU⊆ Graph Search

- No graph version previously existing
- Modifications are similar to **DFS**, except subsumption relationships can change



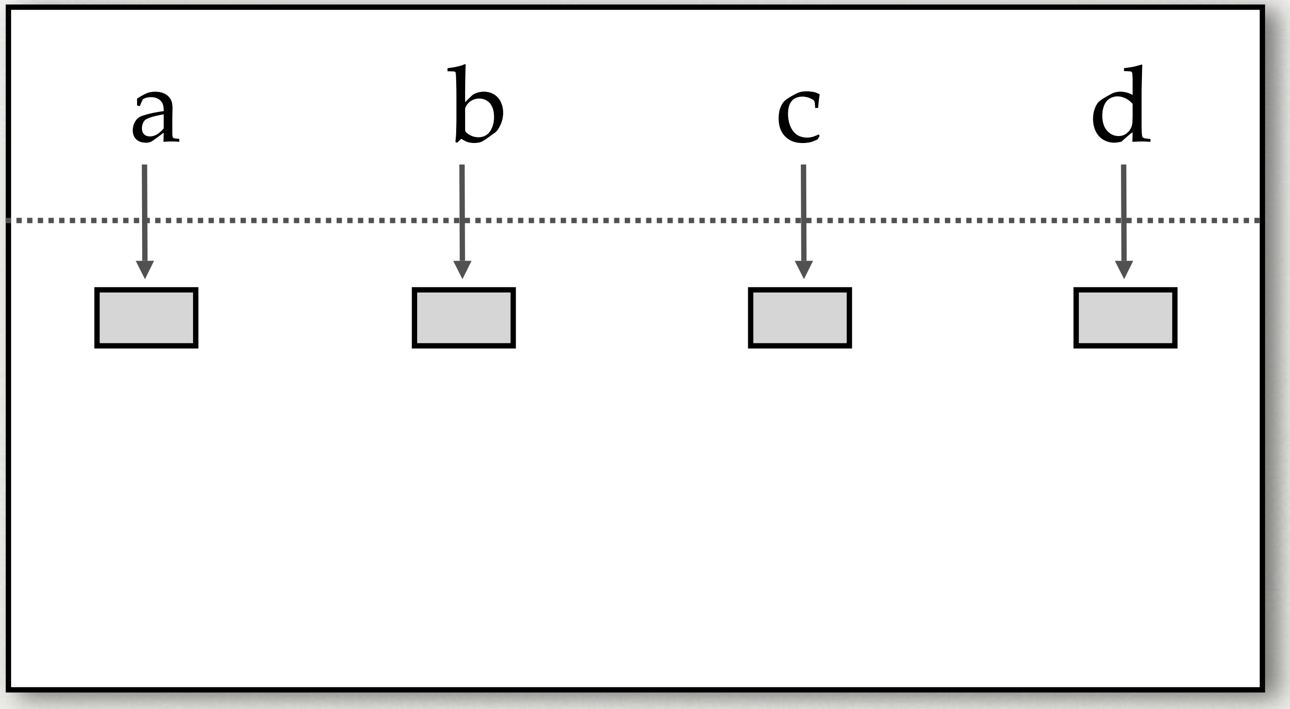



# Finding Related Belief States

- Need data structure that:
  - finds relevant belief states related to query set
  - can efficiently add new sets & change status
  - constant-time lookups, updates not possible?
- We use variant of inverted files, found to work well in many different settings [Helmer and Moerkotte (1999)]

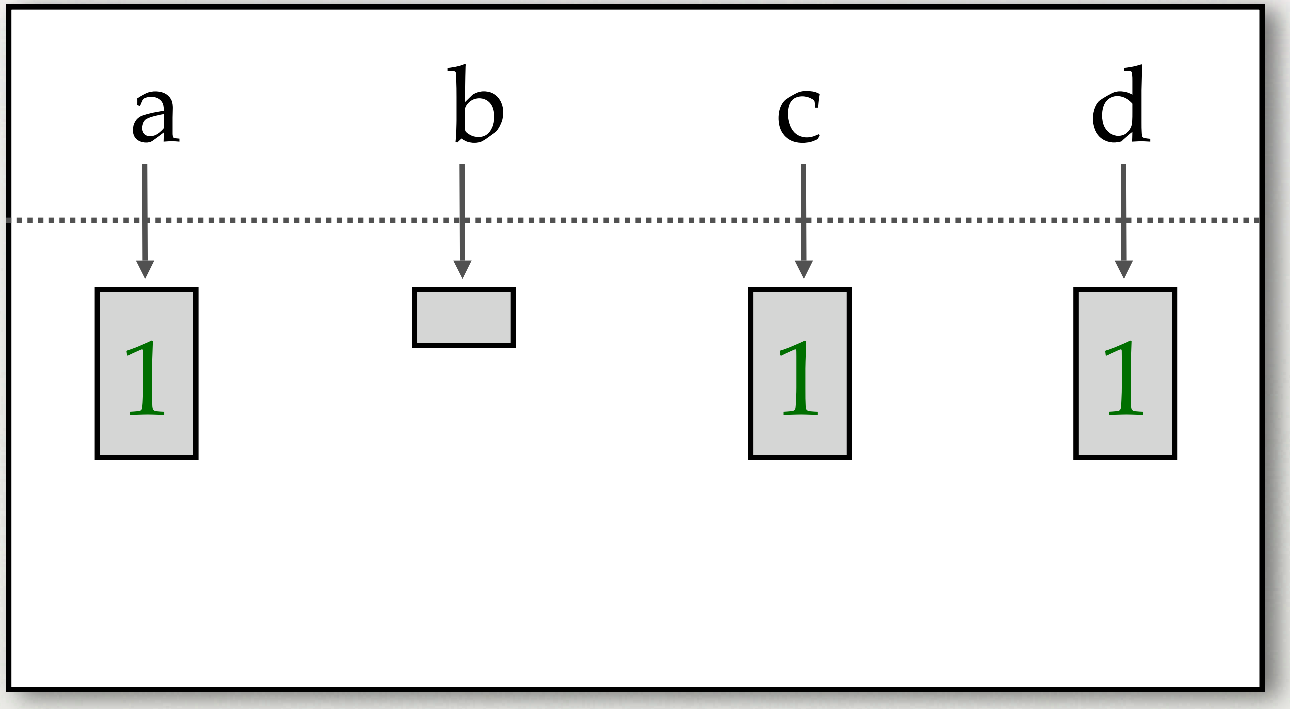
# Inverted Files

- Hash ea. state -> list of belief states it's in



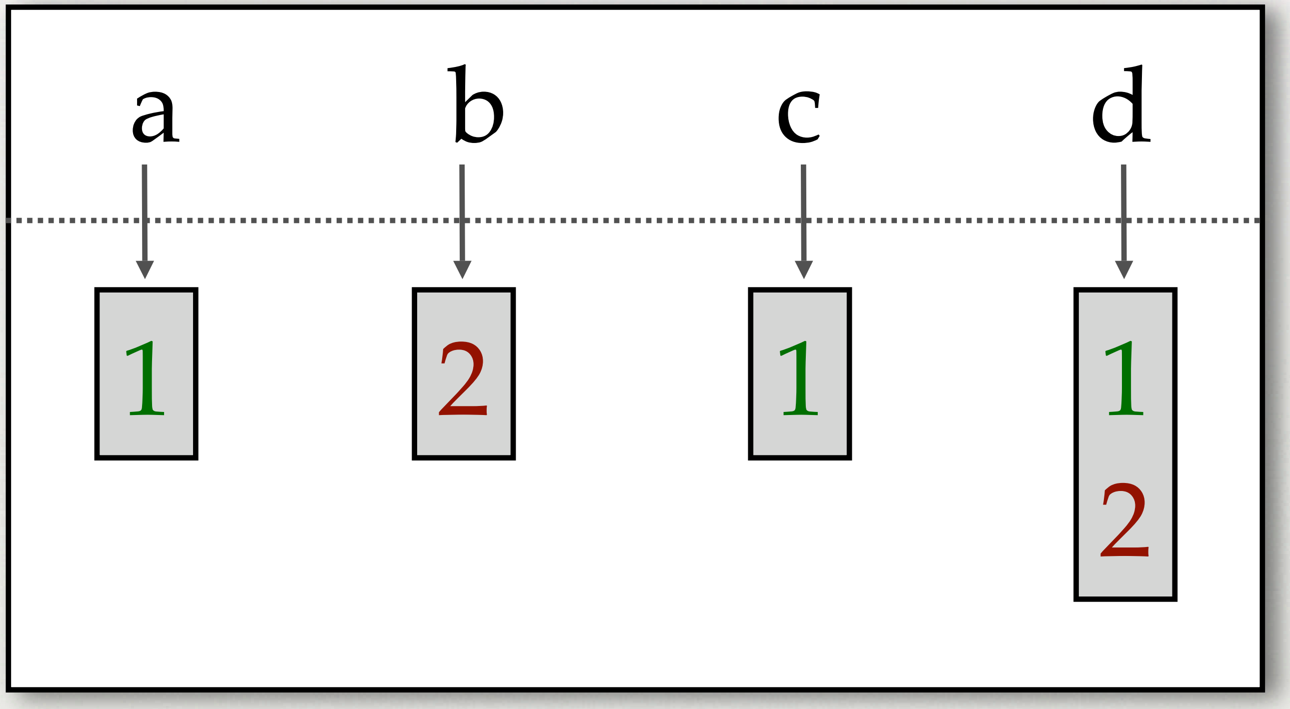
# Inverted Files

- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}



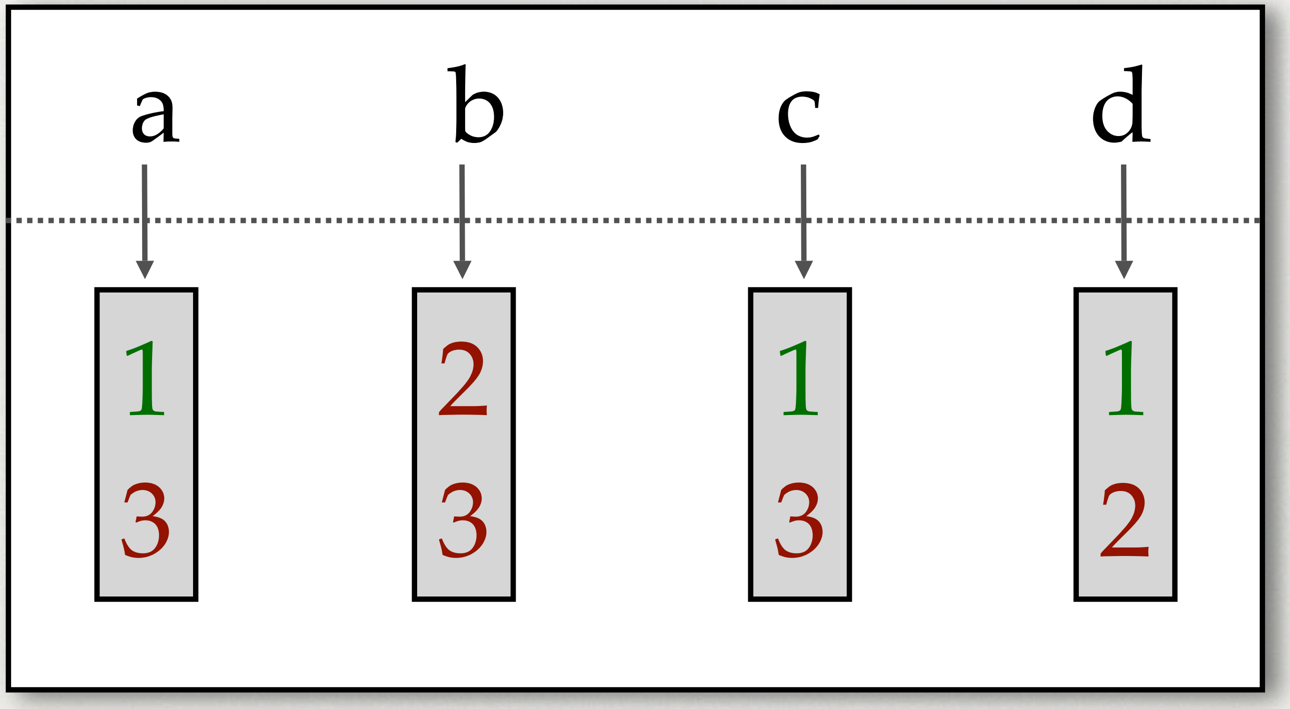
# Inverted Files

- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}



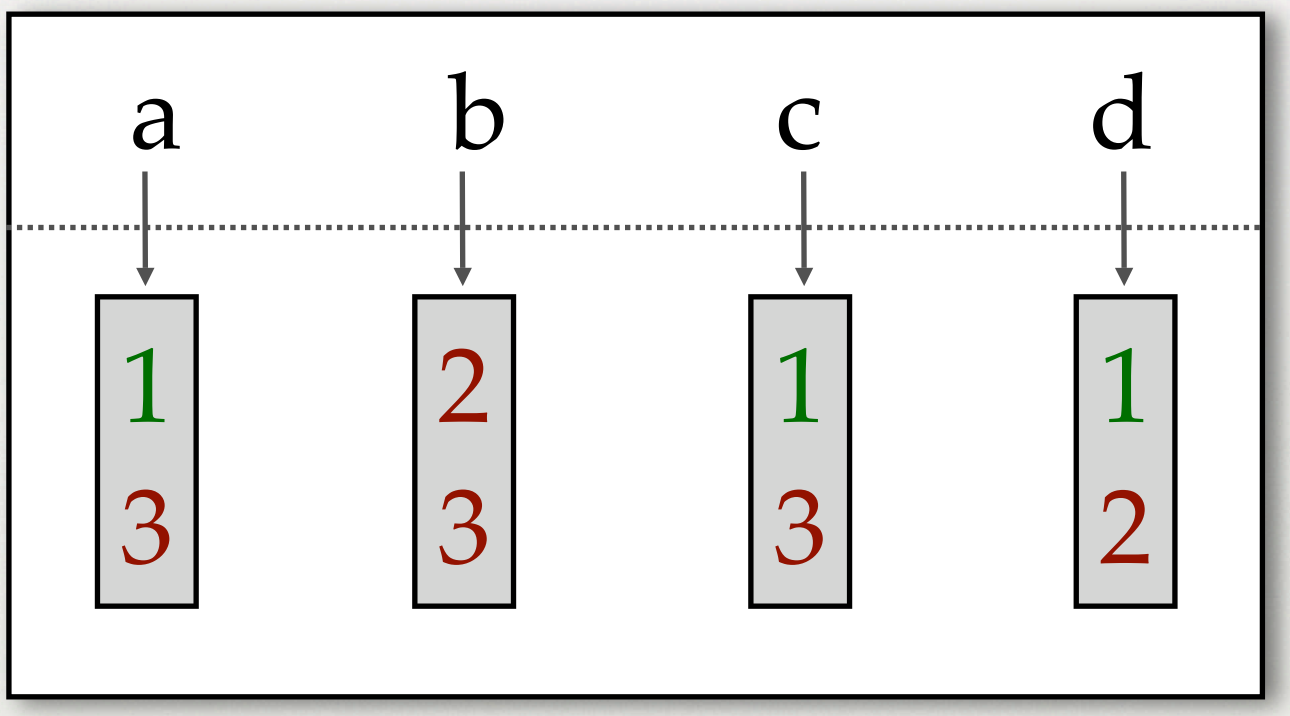
# Inverted Files

- Hash ea. state -> list of belief states it’s in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}



# Inverted Files

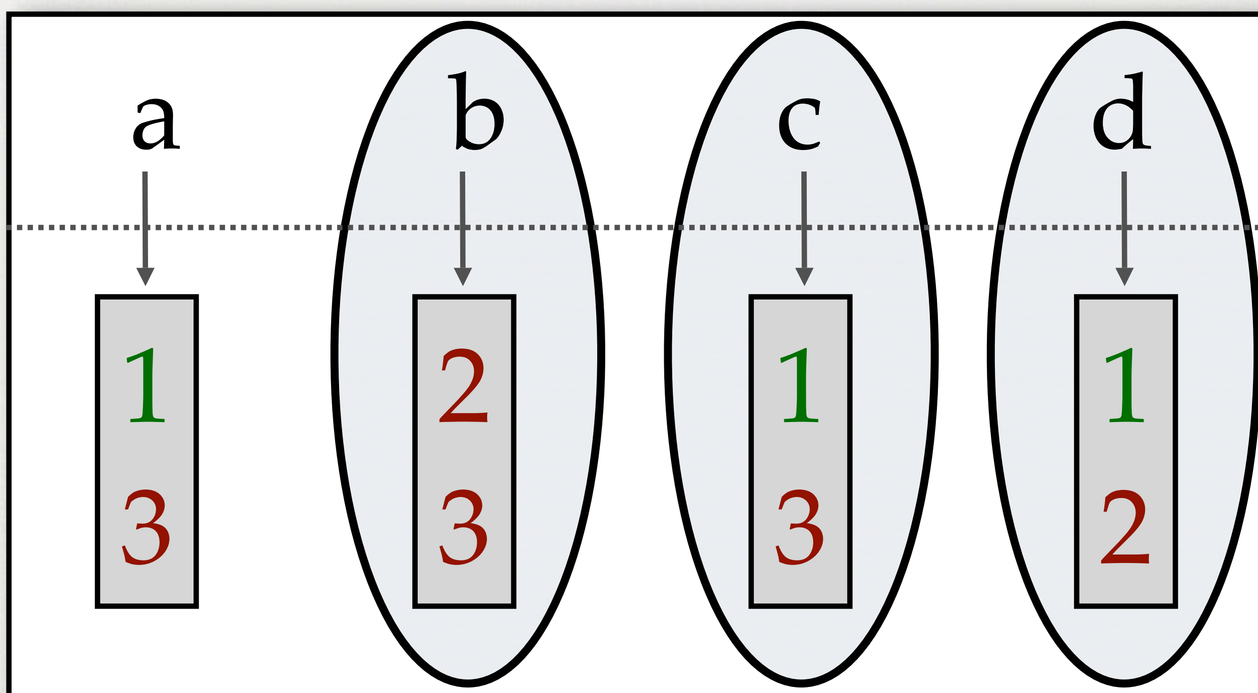
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {b,c,d}

# Inverted Files

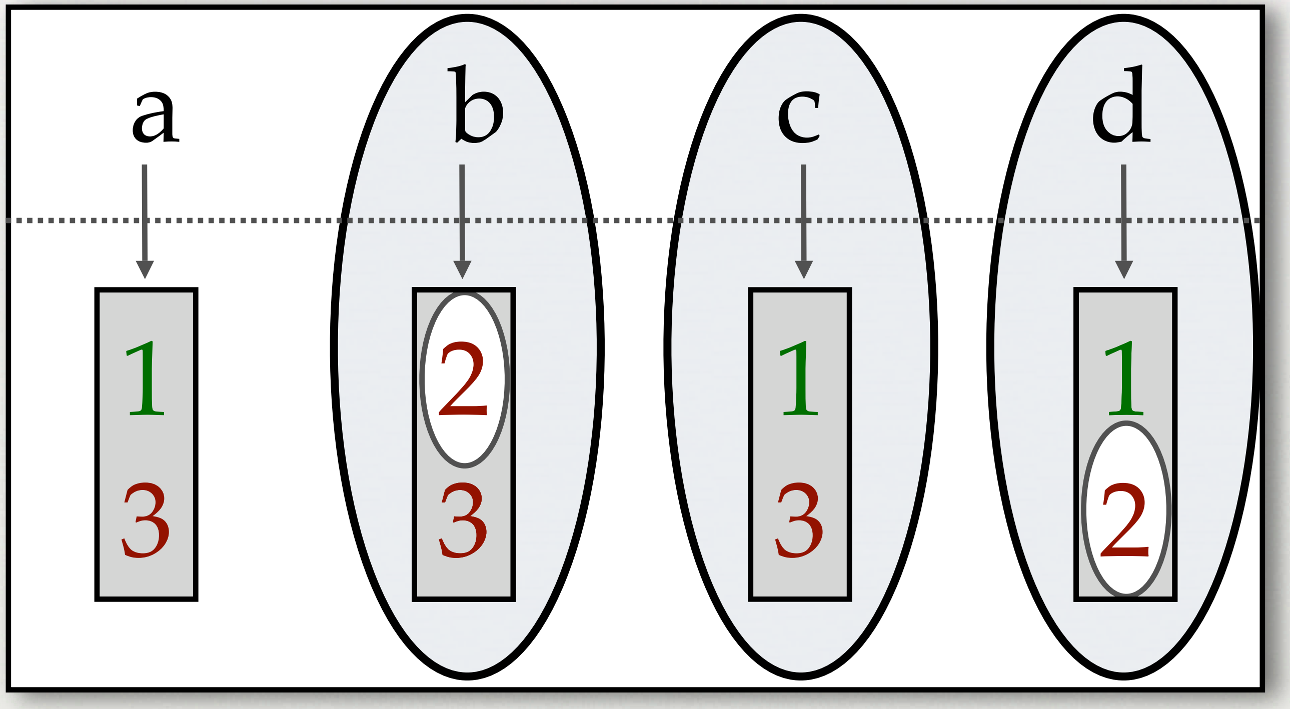
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {b,c,d}

# Inverted Files

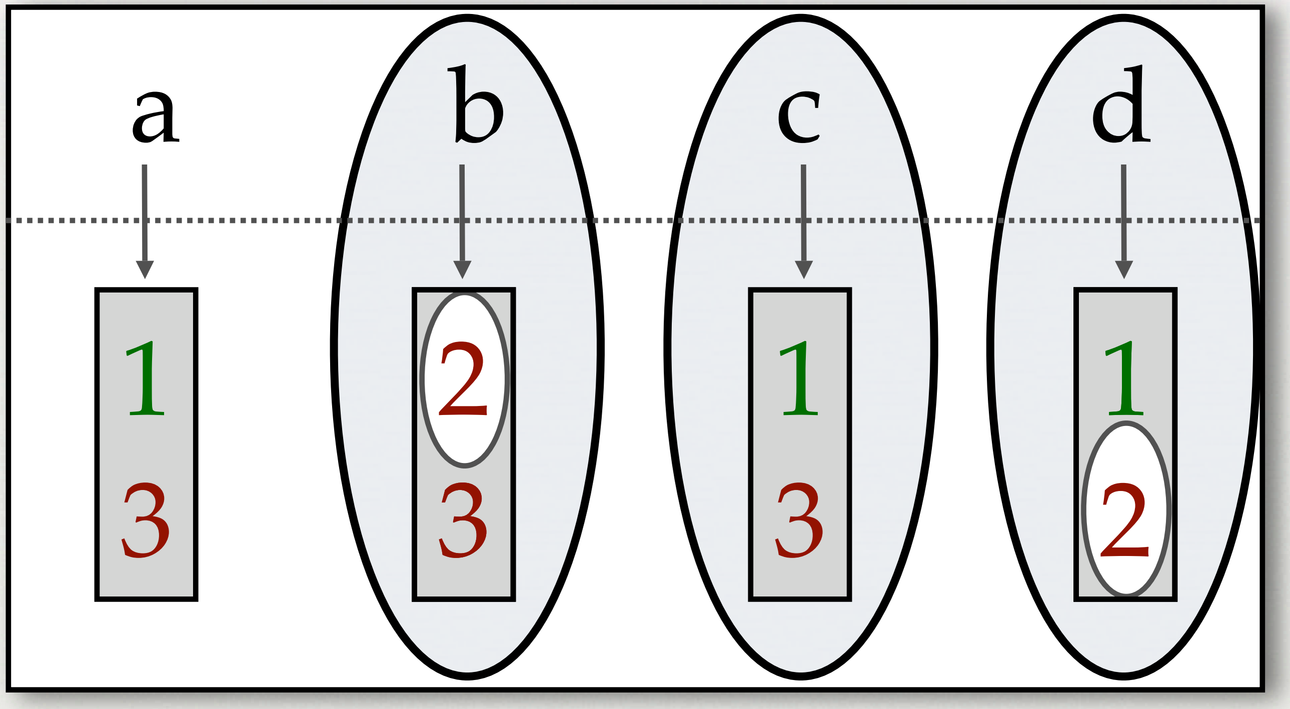
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {b,c,d}

# Inverted Files

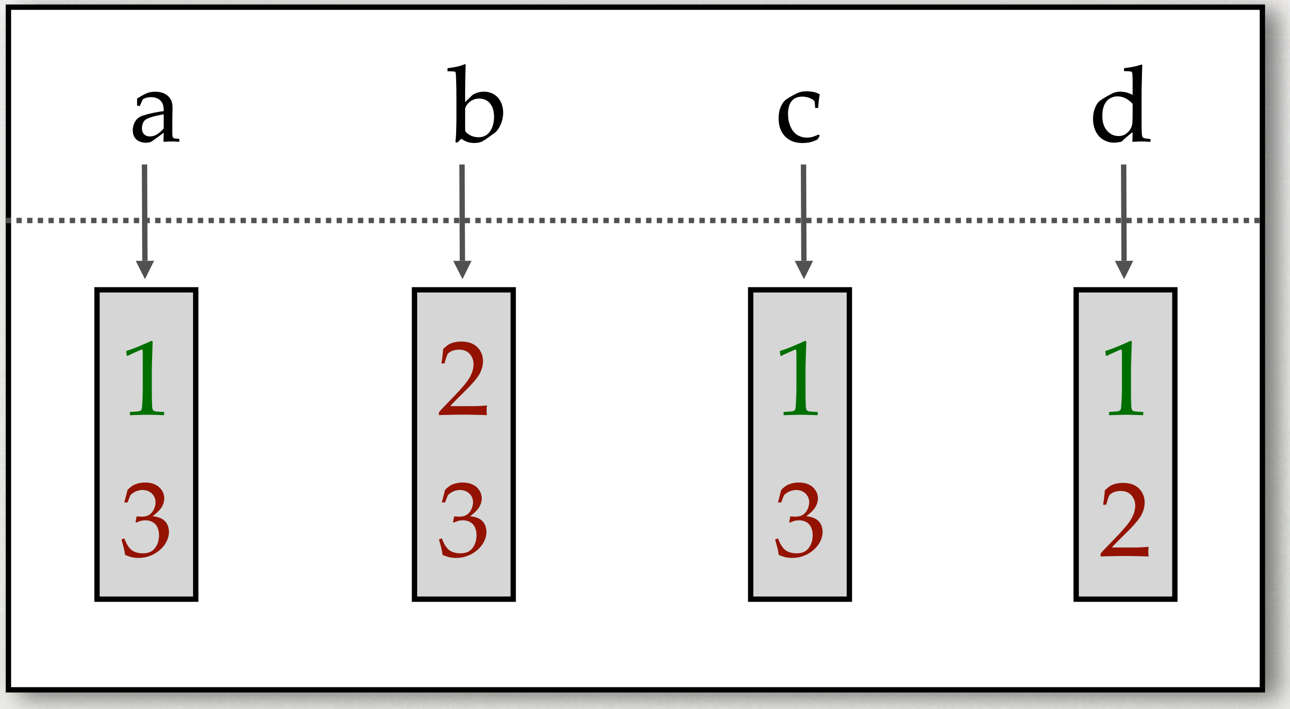
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {b,c,d}

# Inverted Files

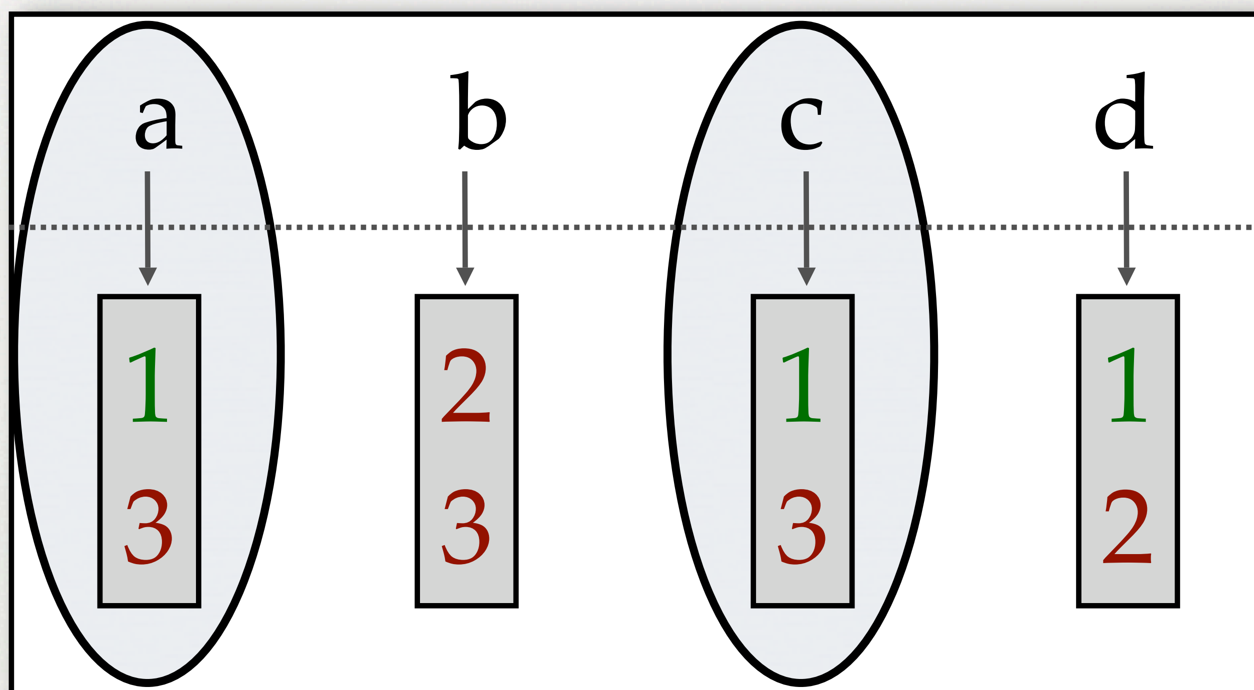
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {a,c}

# Inverted Files

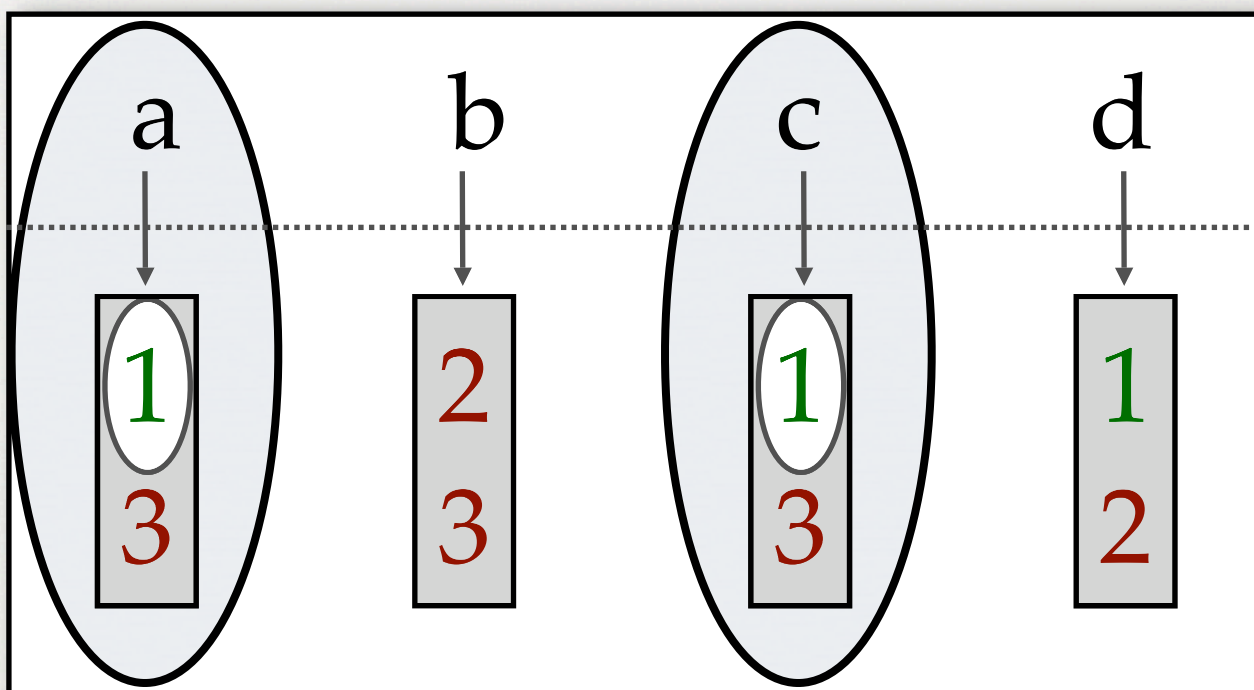
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {a,c}

# Inverted Files

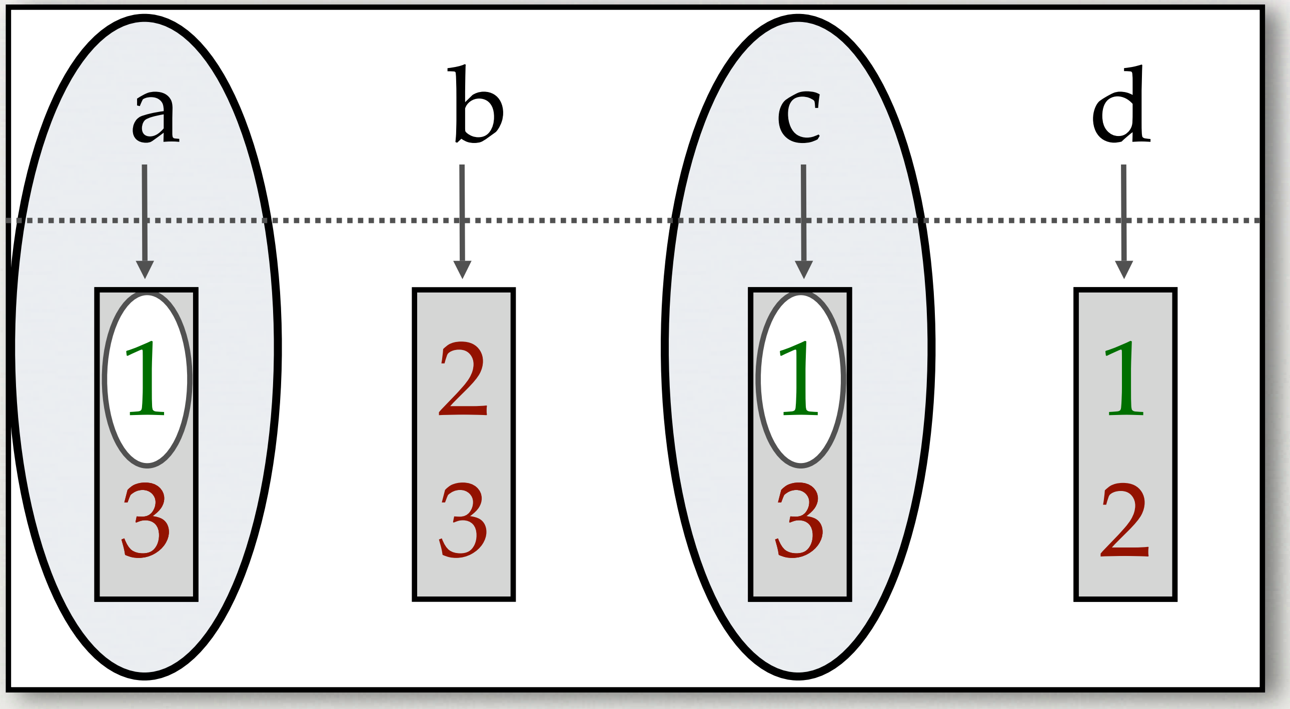
- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {a,c}

# Inverted Files

- Hash ea. state -> list of belief states it's in
- 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}




- Example: query {a,c}

# Experiments

- Two very different domains
- Six total algorithms:
  - **DFS** and **DBU** (tree search)
  - **DFS=** and **DBU=** (hash tables)
  - **DFS⊆** and **DBU⊆** (inverted files)

# Experiments

- Two very different domains
- Six total algorithms:
  - **DFS** and **DBU** (tree search)
  - ****DFS=** and **DBU=** (hash tables)
  - **DFS⊆** and **DBU⊆** (inverted files)

  *Best previous algorithm

# Experiments

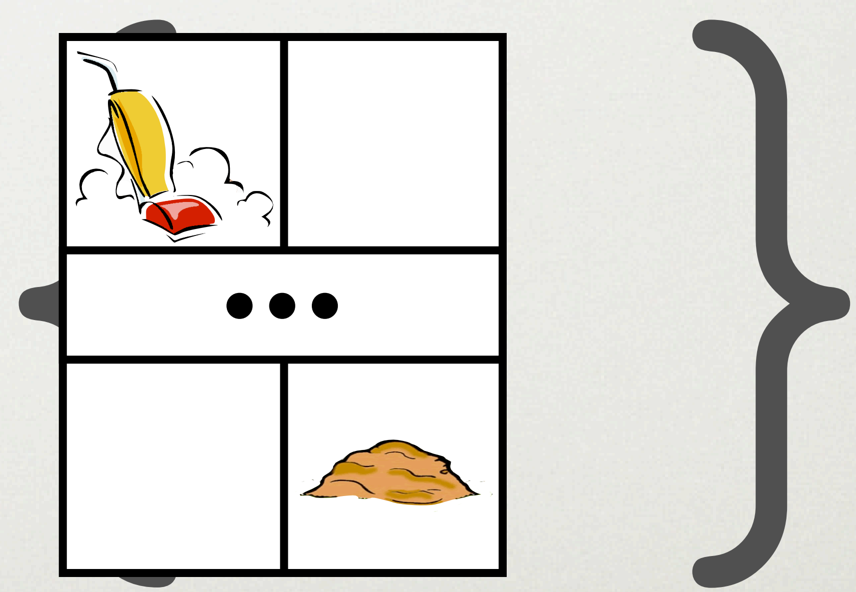
- Two very different domains
- Six total algorithms:
  - **DFS** and **DBU** (tree search)
  - *$^{*}$**DFS=** and $^{◆}$**DBU=** (hash tables)
  - $^{◆}$**DFS⊆** and $^{◆}$**DBU⊆** (inverted files)

  *Best previous algorithm

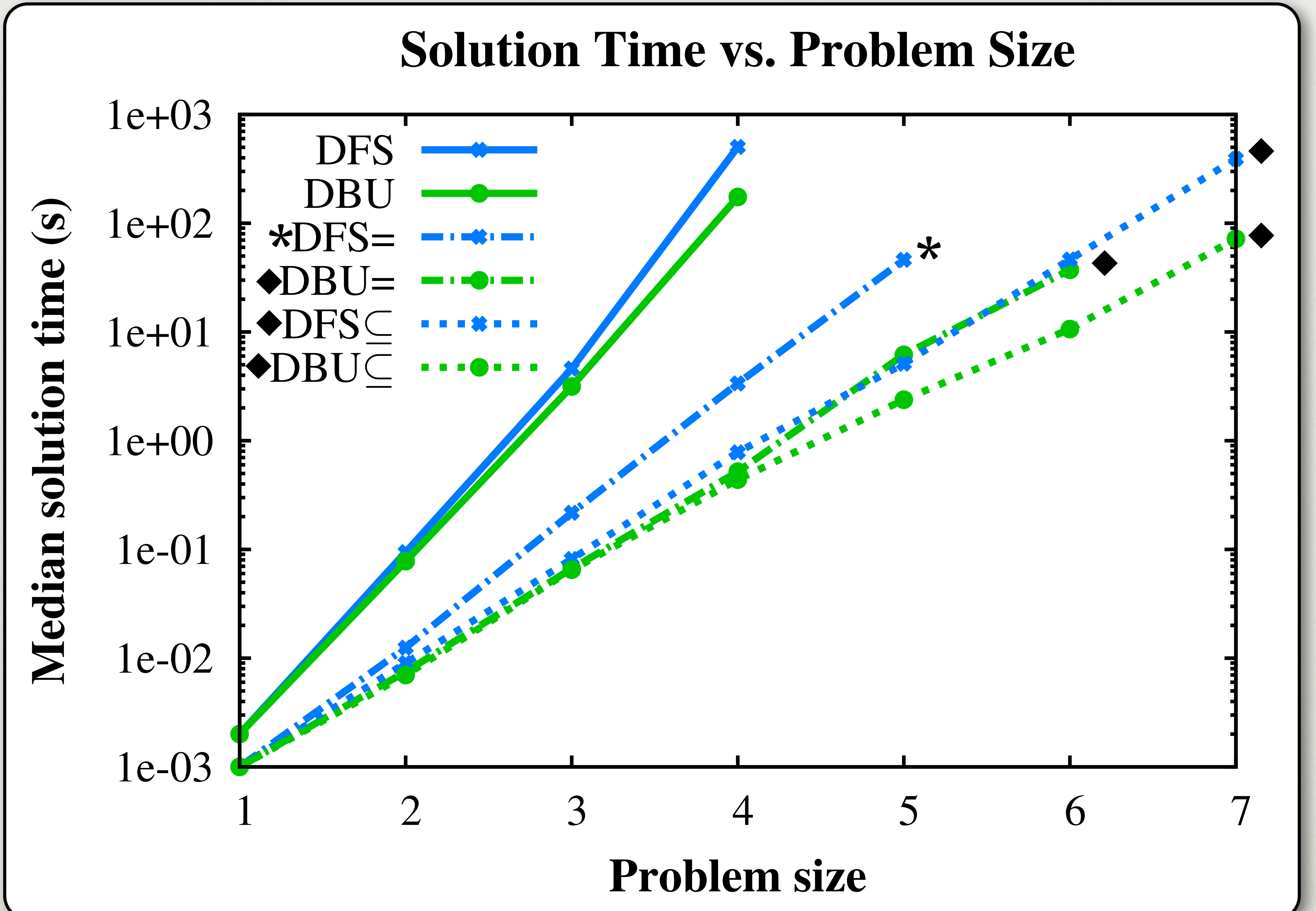
  ◆New algorithm

# Experiments: Vacuum World

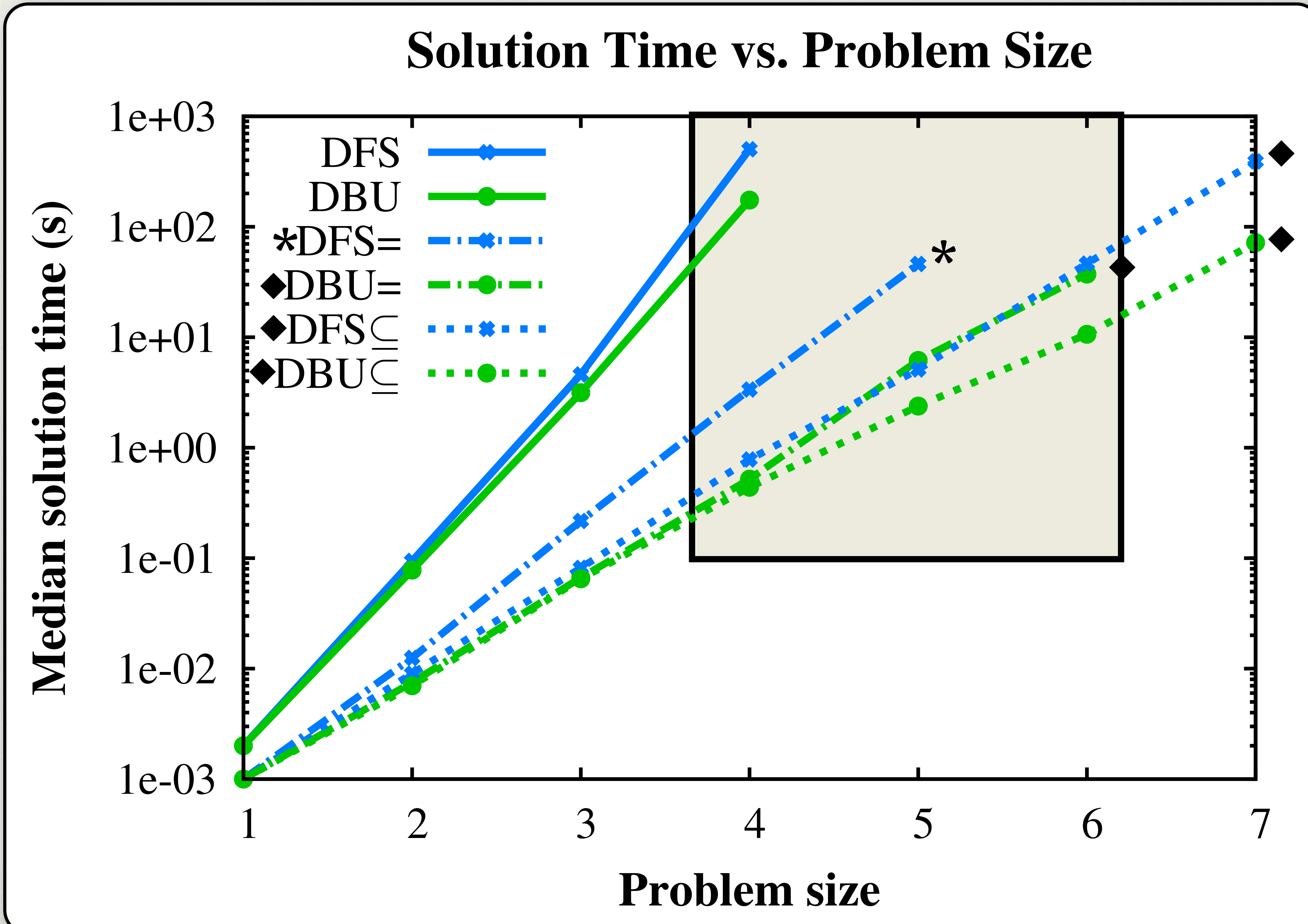
- Domain description:
  - grid world, some squares dirty
  - current square observable
  - actions: *left, right, up, down, suck*
  - *right* & *down* may dirty source square
- Problem instances:
  - world is 2 x N
  - initial belief state =
  - depth limit = 3N+1



# Experiments: Vacuum World



# Experiments: Vacuum World



# Experiments: Vacuum World

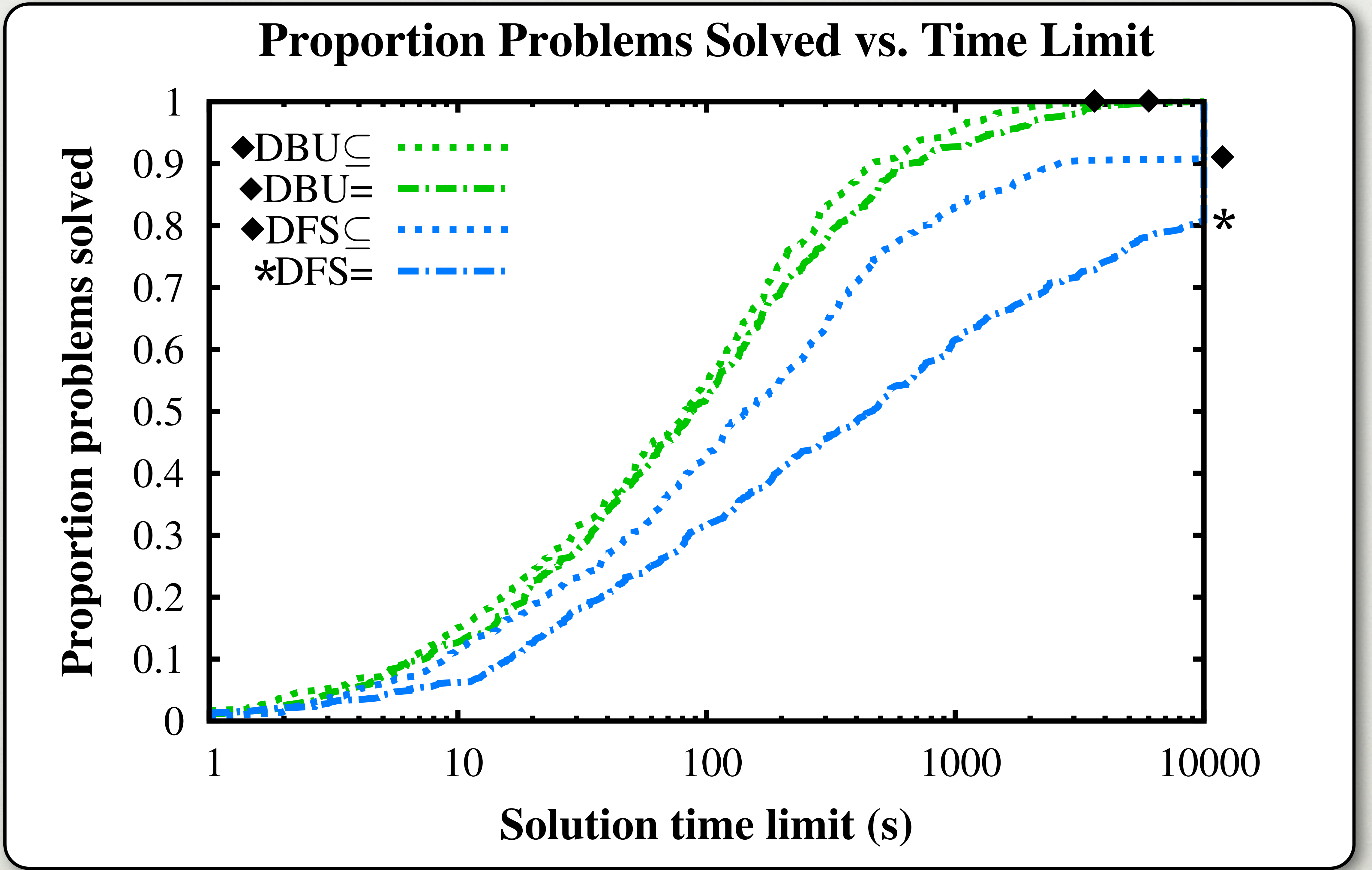
| Size | 2 × 4 | | 2 × 5 | | 2 × 6 | |
|---|---|---|---|---|---|---|
| | Seconds | States | Seconds | States | Seconds | States |
| DFS | 502.3 | 49036K | * | * | * | * |
| DBU | 174.6 | 5892K | * | * | * | * |
| *DFS= | 3.4 | 257K | 46.1 | 3961K | ** | ** |
| ◆DBU= | 0.5 | 11K | 6.2 | 117K | 37.3 | 631K |
| ◆DFS⊆ | 0.8 | 36K | 5.1 | 309K | 46.4 | 3023K |
| ◆DBU⊆ | 0.4 | 10K | 2.4 | 52K | 10.6 | 217K |

* Exceeded 10,000 seconds

** Exceeded 400 MB RAM

# Experiments: Kriegspiel

(7-ply problems; large branching factor)



# Discussion

- >1 order of magnitude speedup over previous algorithms, in 2 domains
  - Subset testing gives big gains for low overhead
- Future work:
  - apply to other algorithms (e.g., PNS)
  - extend to symbolic representations (e.g., BDDs)
  - memory bounded search, garbage collection
  - ...

# Questions?