Maintaining State in Propagation Solvers

Raphael M. Reischuk
IS&C, Saarland University, Germany

Christian Schulte
KTH - Royal Institute of Technology, Sweden

Peter J. Stuckey
NICTA VRL / University of Melbourne, Australia

Guido Tack
PS Lab, Saarland University, Germany

Int'l Conference on Principles and Practice of Constraint Programming

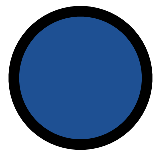
September 20th, 2009, Lisbon, Portugal

Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state

Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



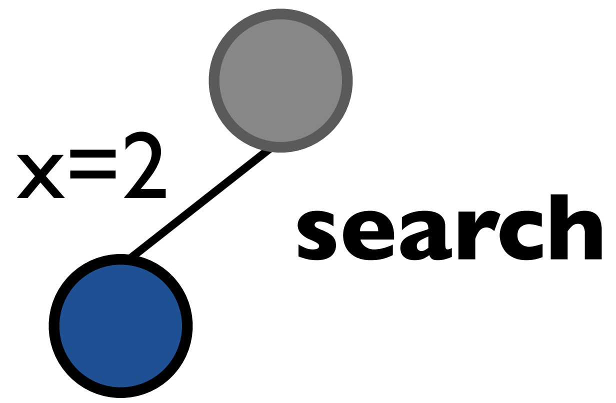
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state

● **propagate**

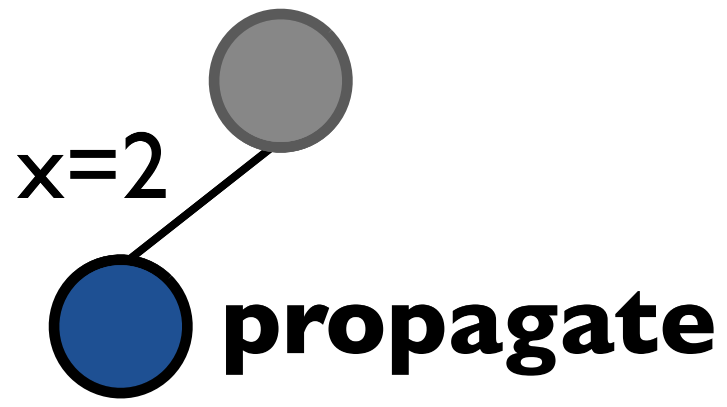
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



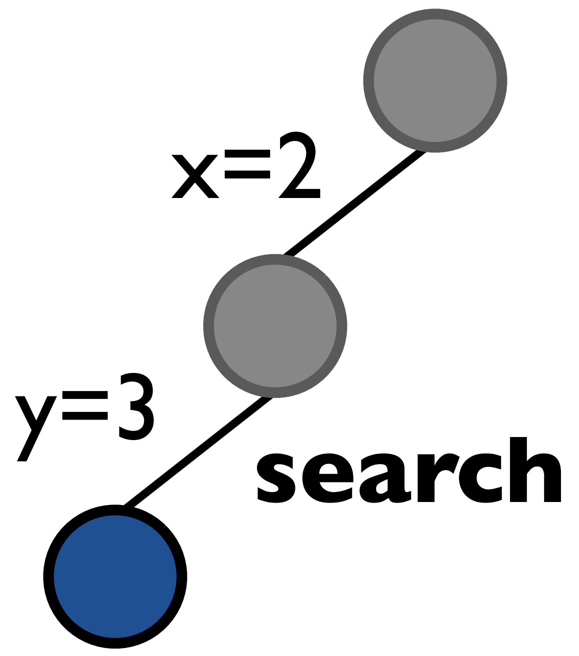
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



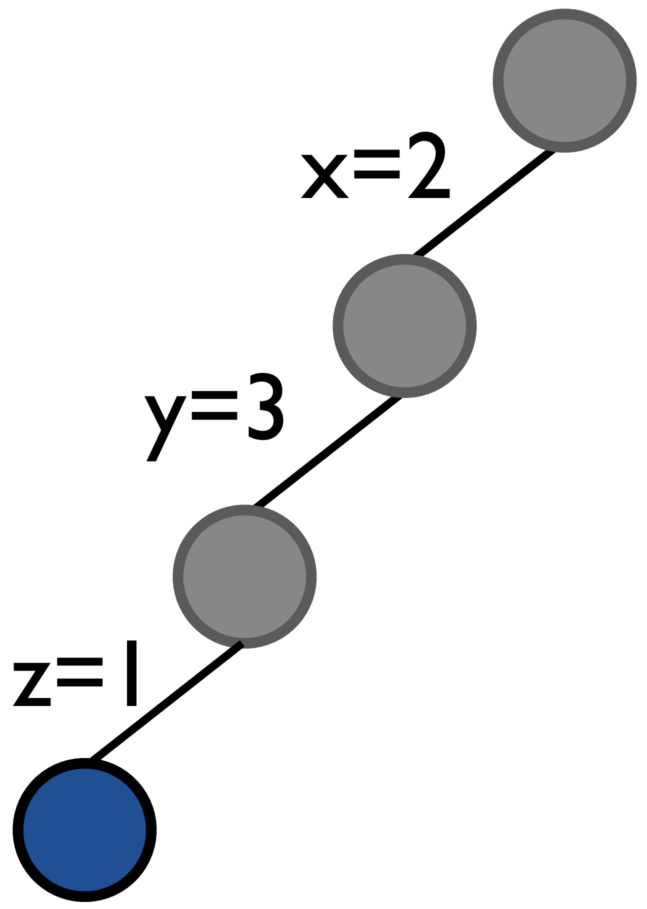
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



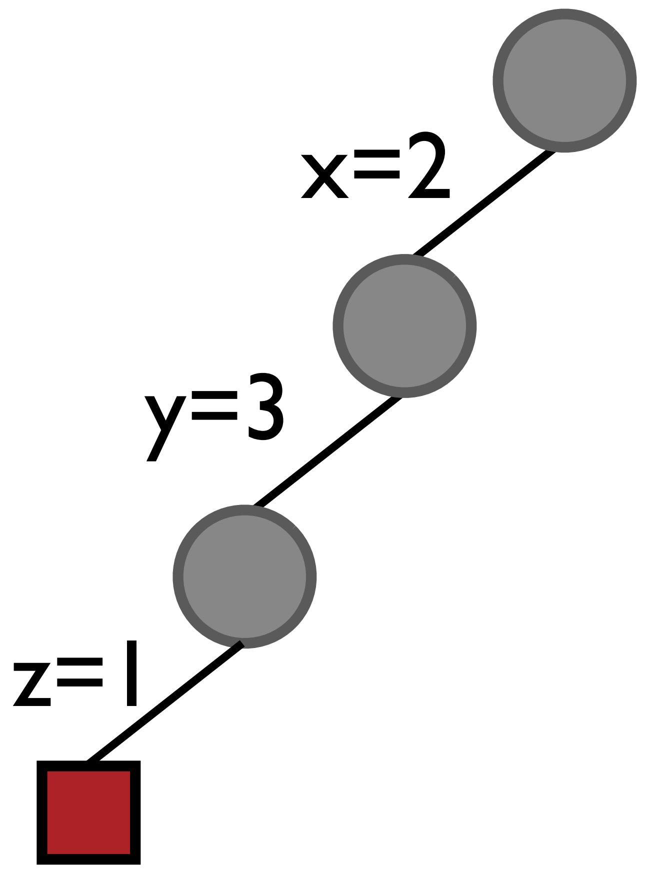
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



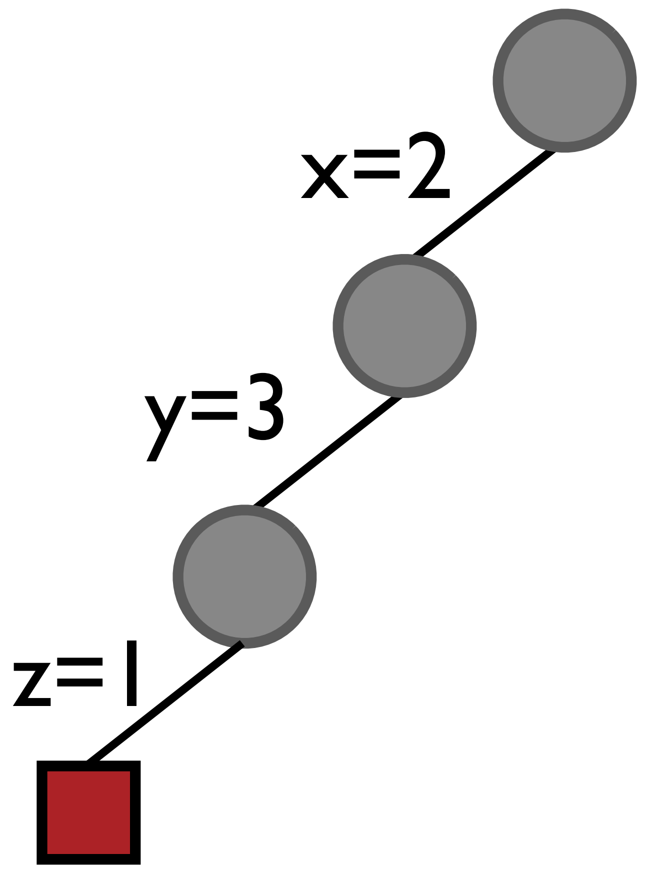
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



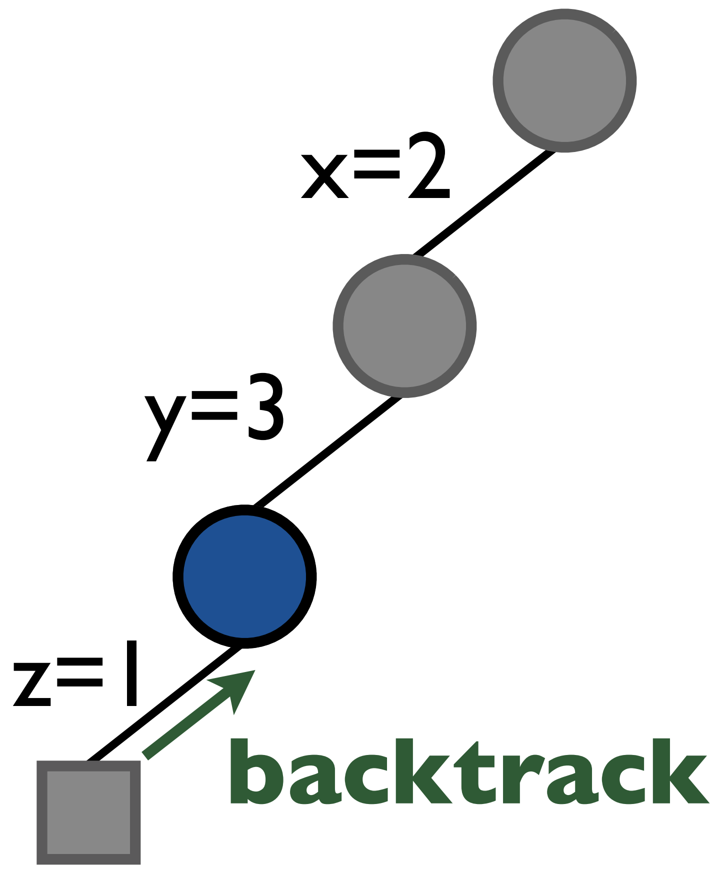
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



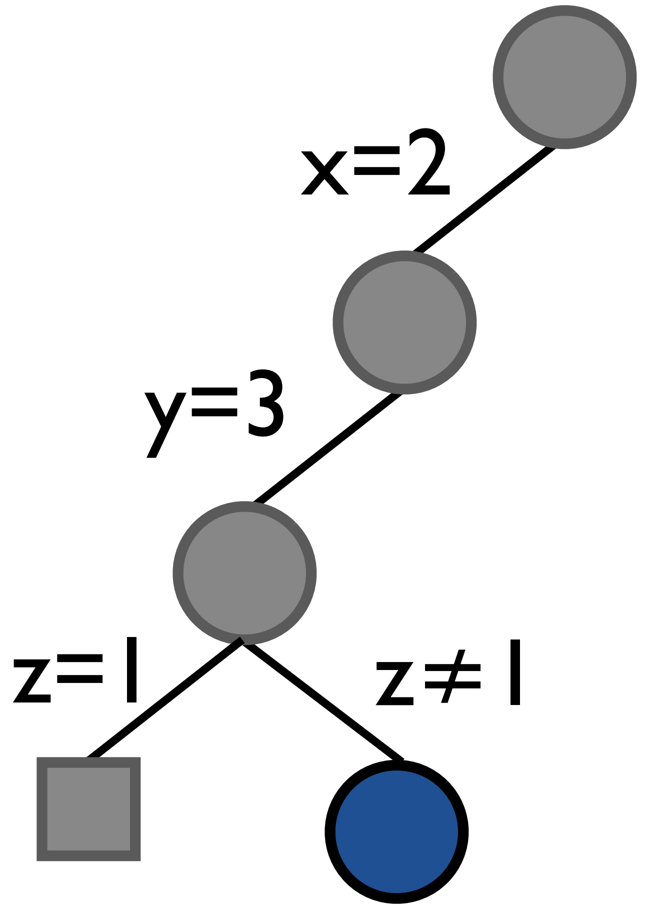
Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



Overview

- Propagation + search **destructively** update state
- Backtracking **recovers** previous (or equivalent) state



Two Goals

- **Survey**
  - what are the stateful data structures?
    (domains, dependencies, internal propagator state)
  - how is state managed during search?
    (trailing, copying, recomputation, static/backtrack-safe state)
- **Evaluation**
  - which state management is the best?
  - how do trailing and recomputation compare?
  - ...in a state-of-the-art system

Survey

Stateful Objects

Stateful Objects

Domain

Dependencies

Propagator

Control

Stateful Objects

Domain	allowed values for variables
Dependencies	which propagator to run when domain changes
Propagator	implements propagation algorithm
Control	queue, trail, search stack

Stateful Objects

Domain	allowed values for variables
Dependencies	which propagator to run when domain changes
Propagator	implements propagation algorithm

Stateful Objects

Domain	always stateful	allowed values for variables
Dependencies		which propagator to run when domain changes
Propagator		implements propagation algorithm

Stateful Objects

Domain	always stateful	allowed values for variables
Dependencies	watched literals, entailed propagators	which propagator to run when domain changes
Propagator		implements propagation algorithm

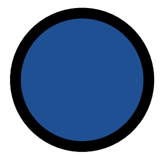
Stateful Objects

Domain	always stateful	allowed values for variables
Dependencies	watched literals, entailed propagators	which propagator to run when domain changes
Propagator	incrementality	implements propagation algorithm

Naive Search

- Copy state for both children
- Naive because copying takes time and memory
- Goal:

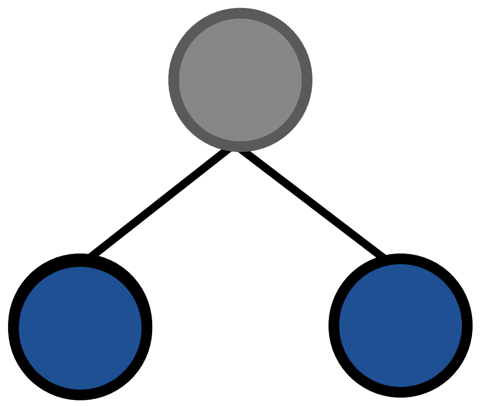
  get rid of copies



Naive Search

- Copy state for both children
- Naive because copying takes time and memory
- Goal:

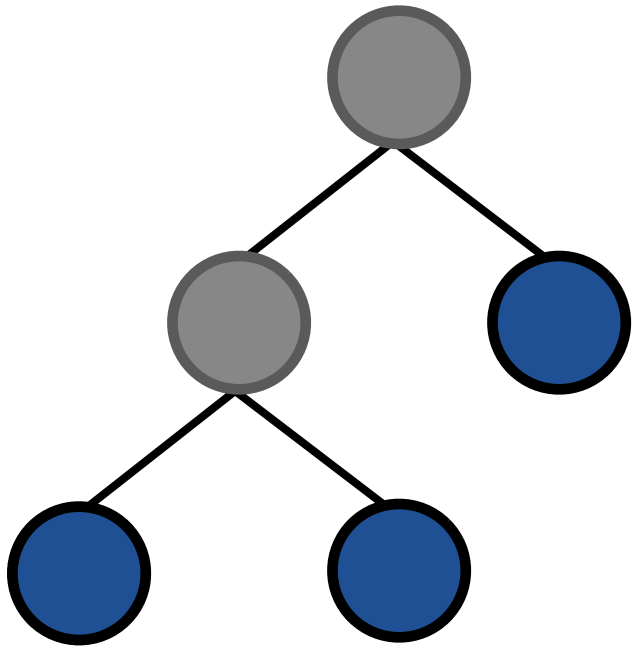
  get rid of copies



Naive Search

- Copy state for both children
- Naive because copying takes time and memory
- Goal:

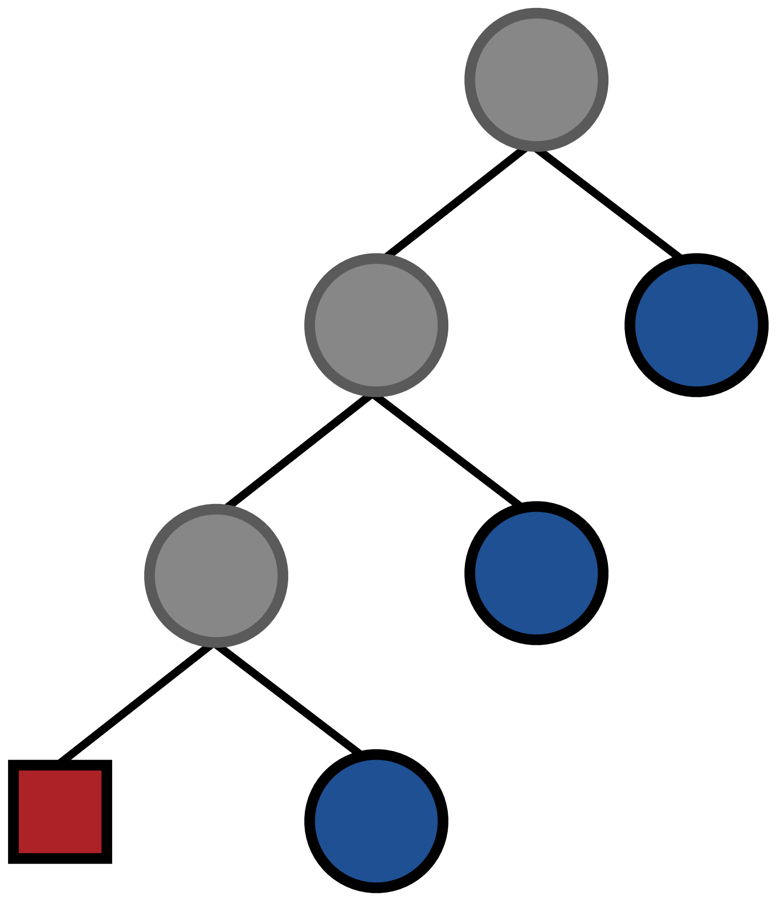
  get rid of copies



Naive Search

- Copy state for both children
- Naive because copying takes time and memory
- Goal:

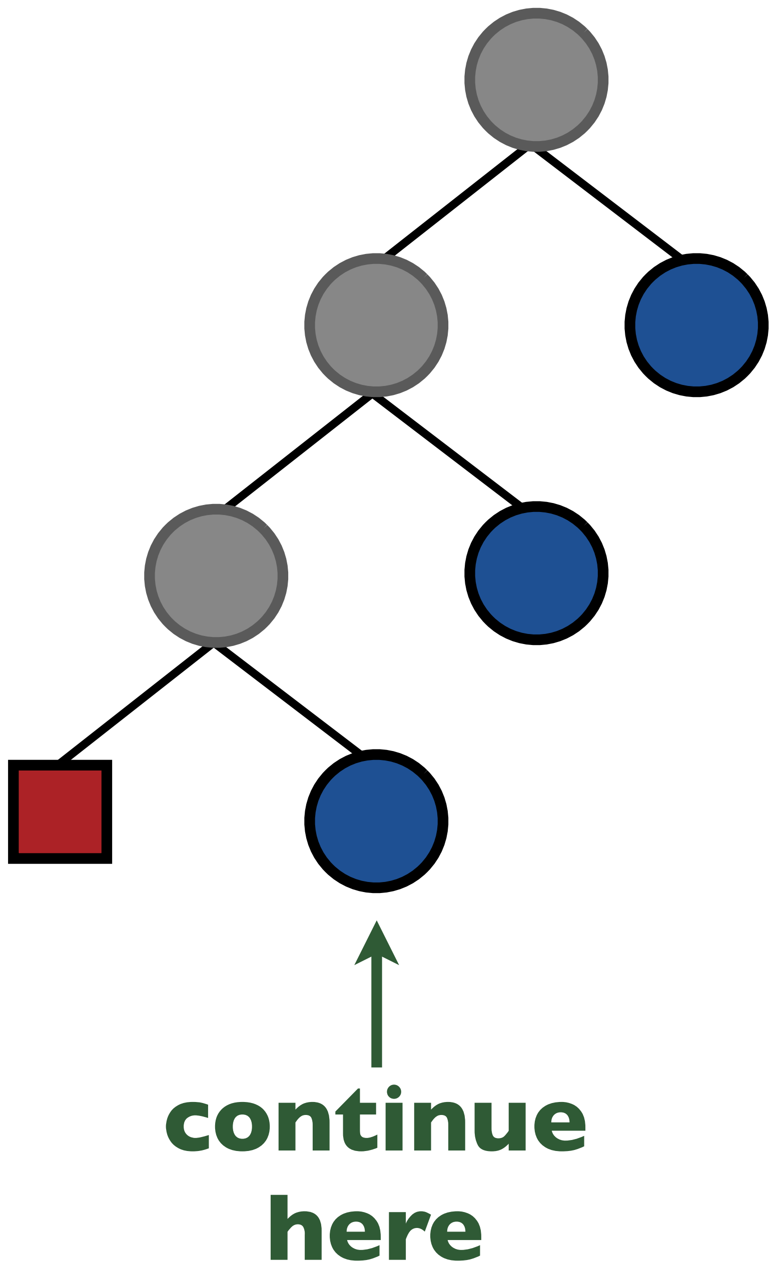
  get rid of copies



Naive Search

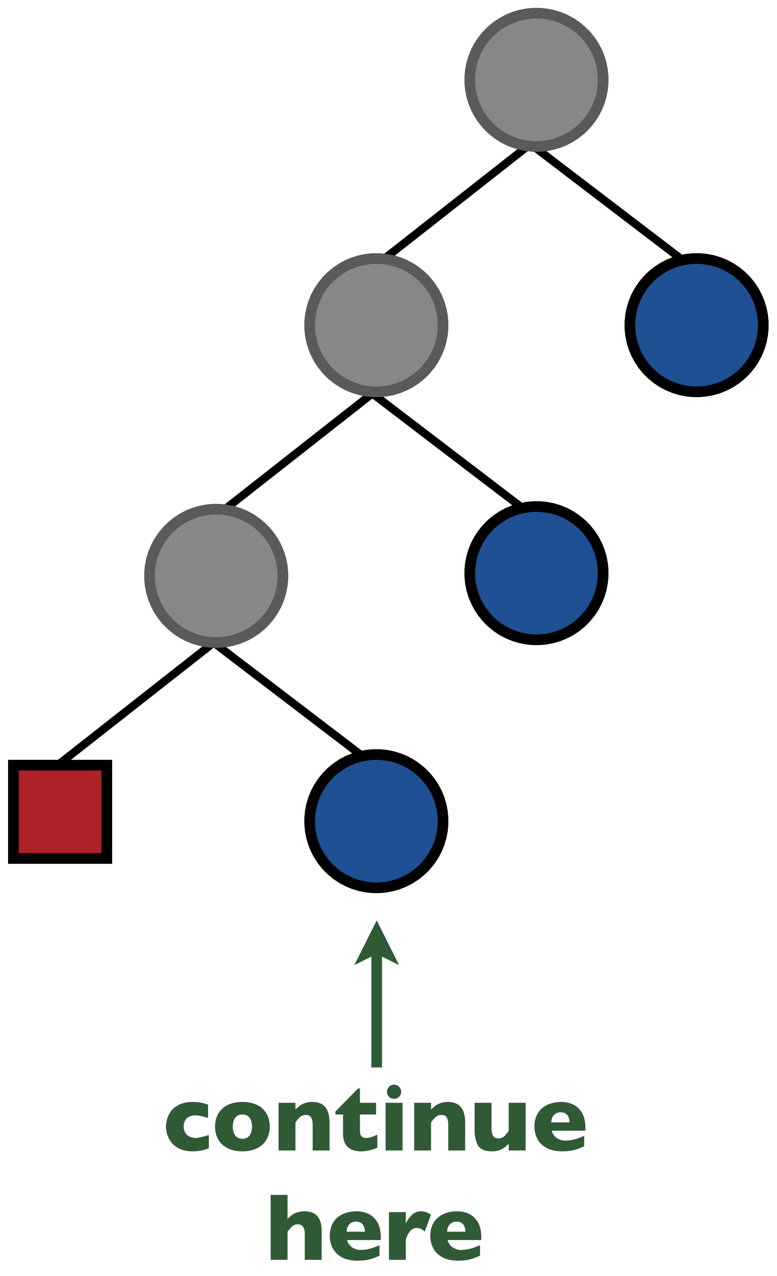
- Copy state for both children
- Naive because copying takes time and memory
- Goal:

  get rid of copies



Naive Search

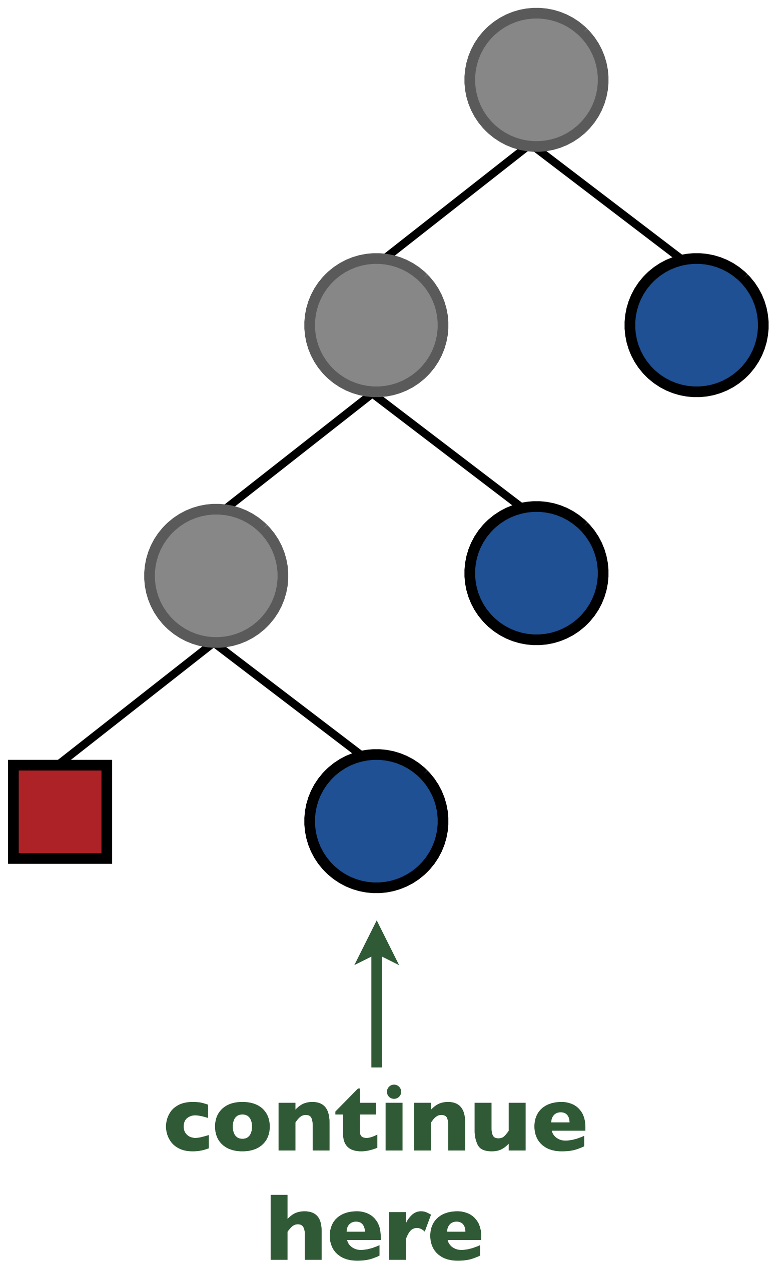
- Copy state for both children
- Naive because copying takes time and memory



Naive Search

- Copy state for both children
- Naive because copying takes time and memory
- Goal:

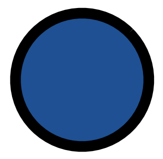
  get rid of copies



Global State

- **One** global state
- Maintained by **trailing**

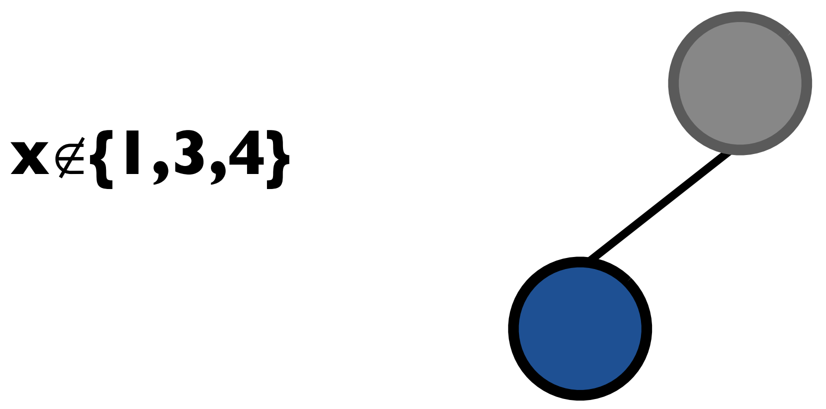
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

  or **static/backtrack-safe**



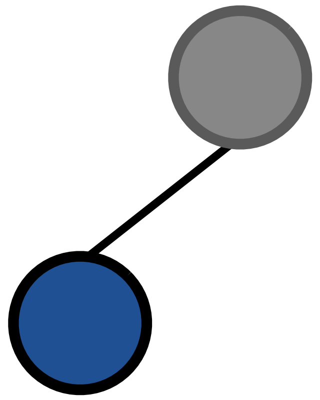
Global State

- **One** global state
- Maintained by **trailing**

  or **static/backtrack-safe**

$x \notin \{1,3,4\}$

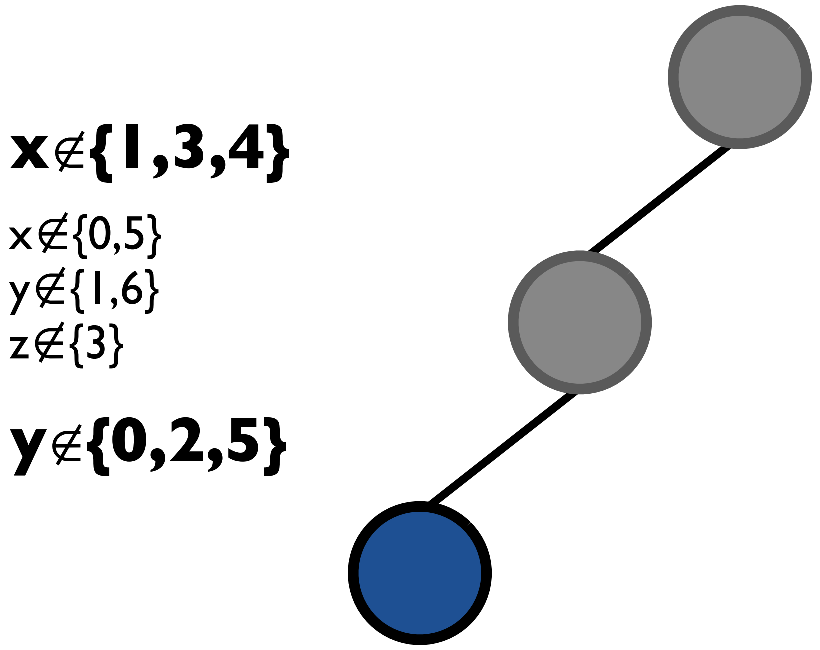
$x \notin \{0,5\}$
$y \notin \{1,6\}$
$z \notin \{3\}$



Global State

- **One** global state
- Maintained by **trailing**

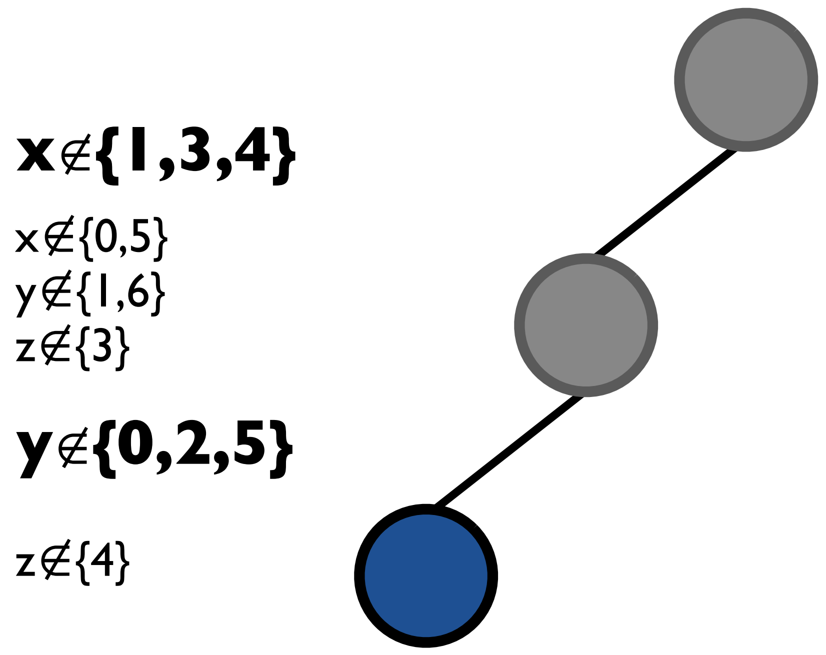
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

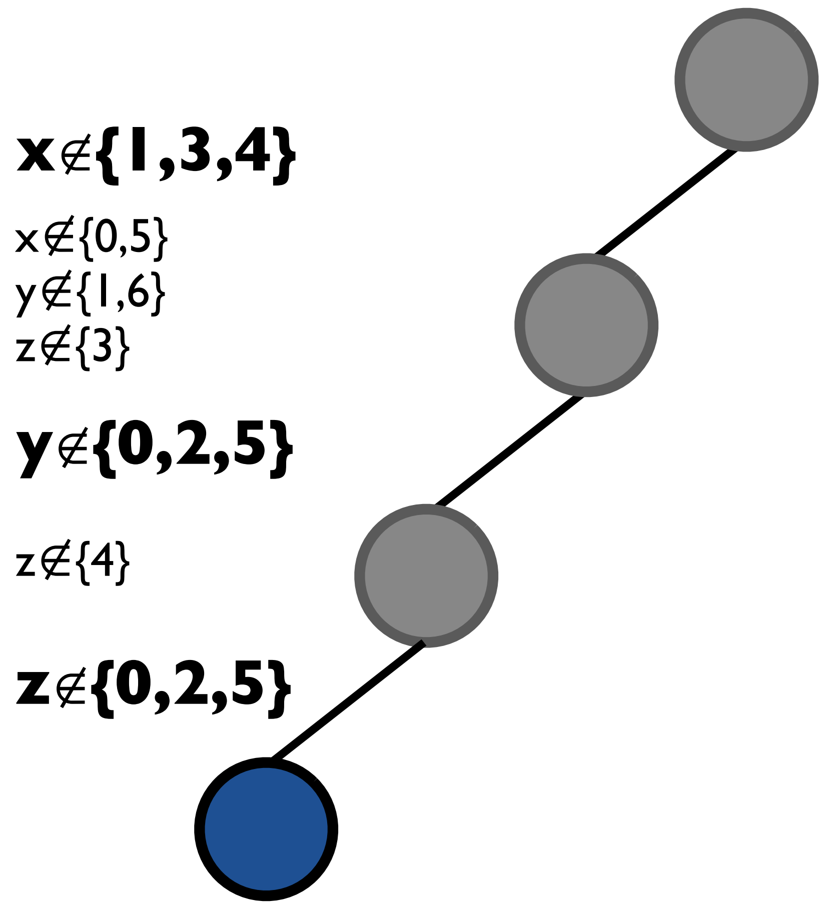
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

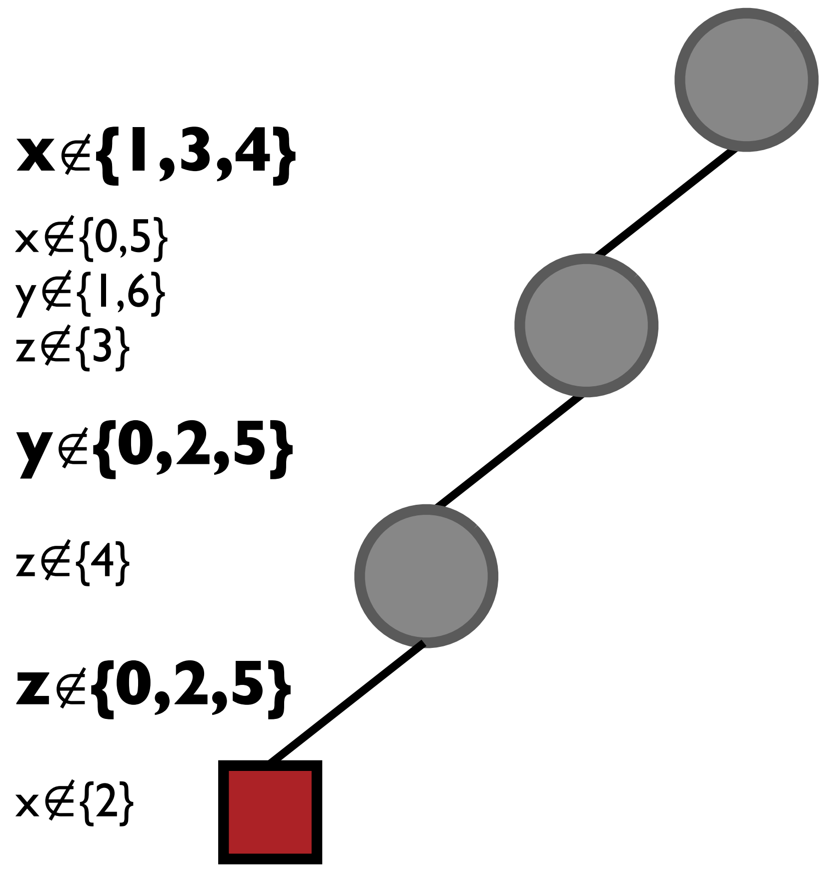
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

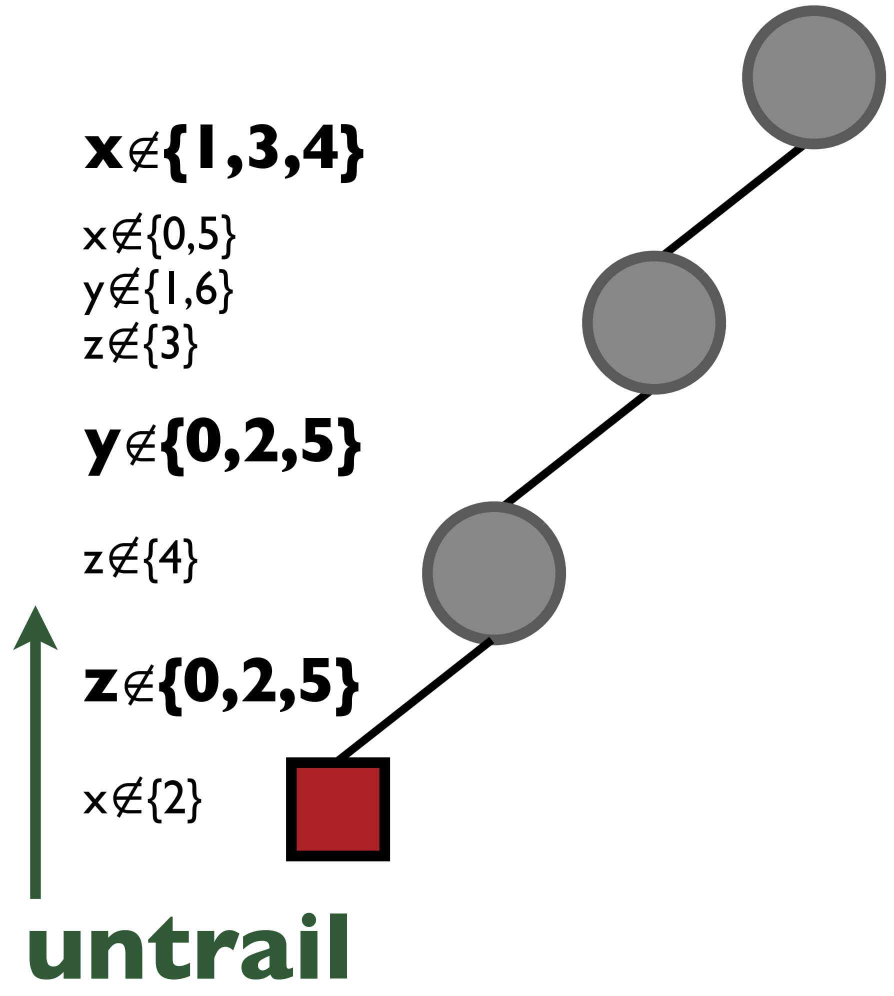
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

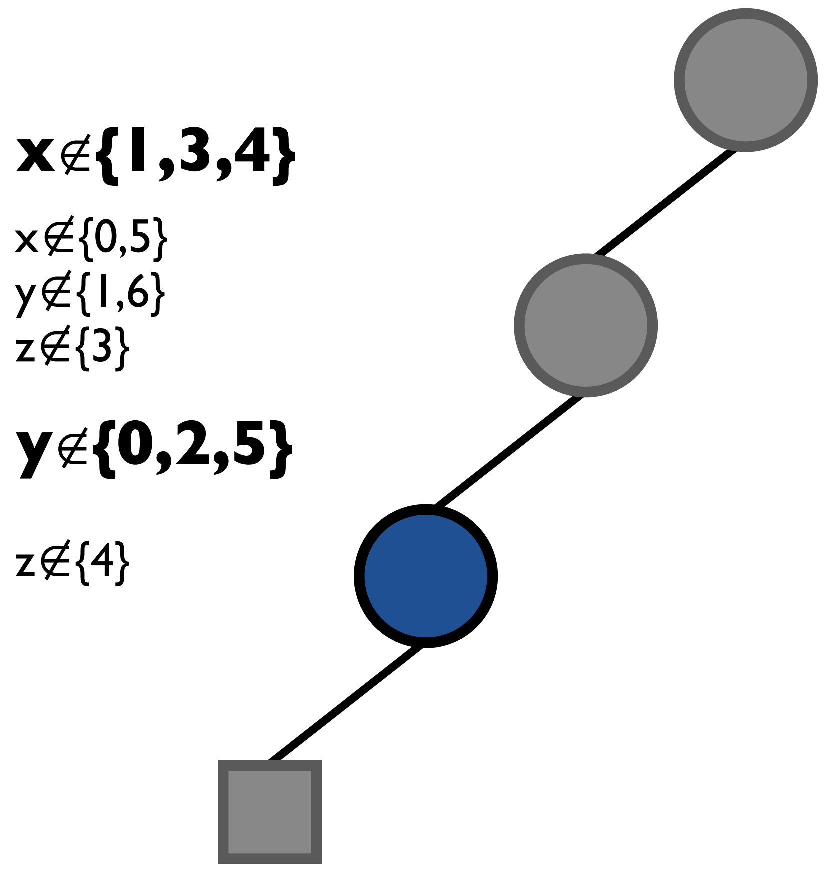
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

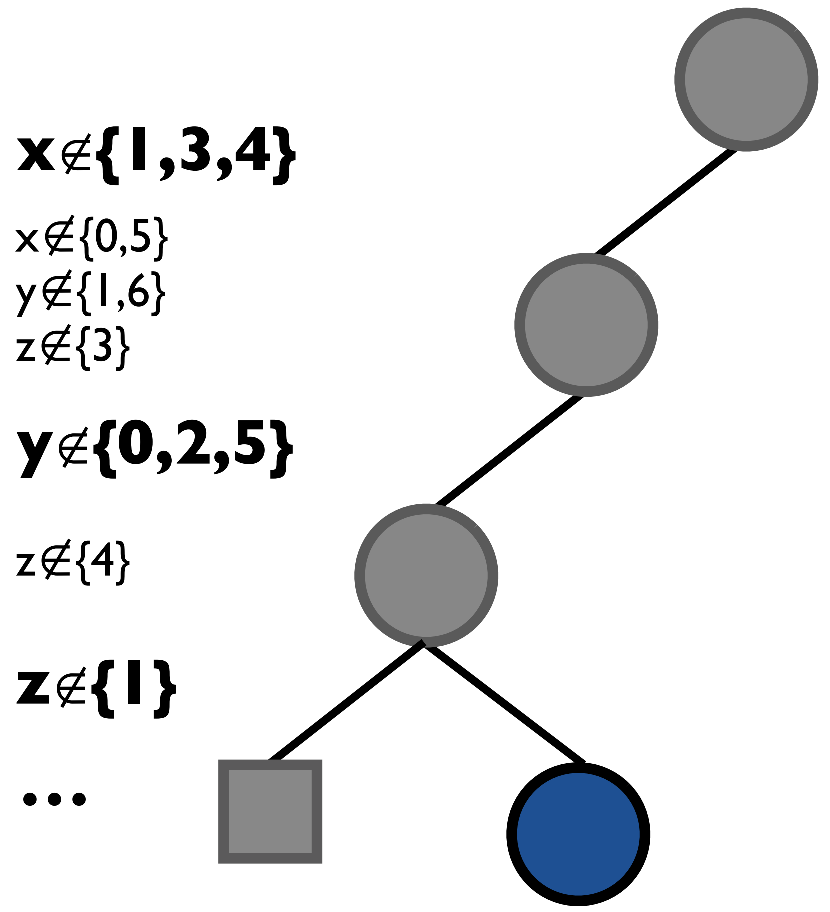
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

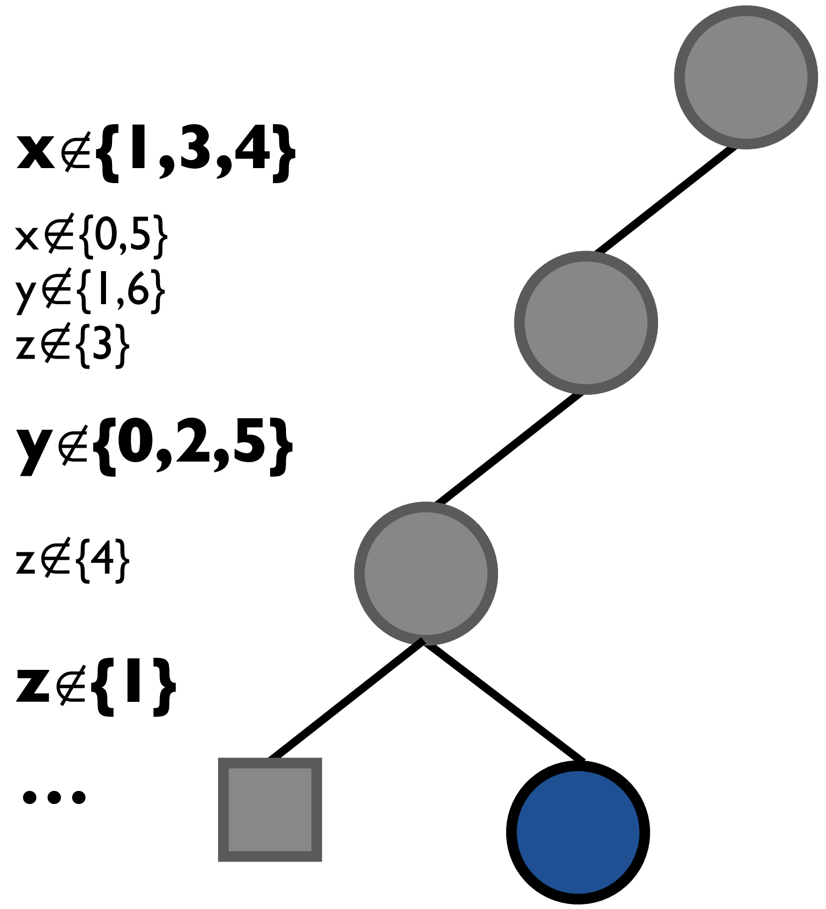
  or **static/backtrack-safe**



Global State

- **One** global state
- Maintained by **trailing**

  or **static/backtrack-safe**


$\mathbf{x \notin \{1,3,4\}}$
$x \notin \{0,5\}$
$y \notin \{1,6\}$
$z \notin \{3\}$
$\mathbf{y \notin \{0,2,5\}}$
$z \notin \{4\}$
$\mathbf{z \notin \{1\}}$
…

Static / backtrack-safe State

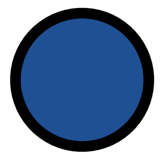
- **Static state:** never modified during search
  - e.g. DFA for regular constraint, tuple sets for extensional constraints, arrays of coefficients for linear constraints etc.

Static / backtrack-safe State

- **Static state:** never modified during search
  - e.g. DFA for regular constraint, tuple sets for extensional constraints, arrays of coefficients for linear constraints etc.
- **Backtrack-safe state:** modifications must be *valid* on path to root node
  - strict DFS backtracking keeps state valid
  - prime example: watched literals (backtrack-safe dependencies)

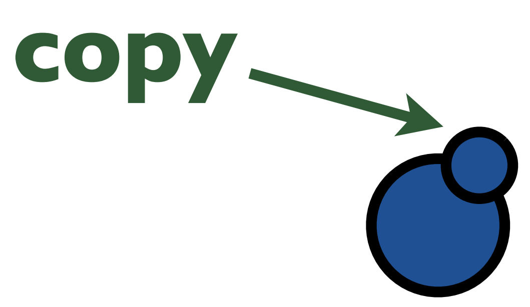
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



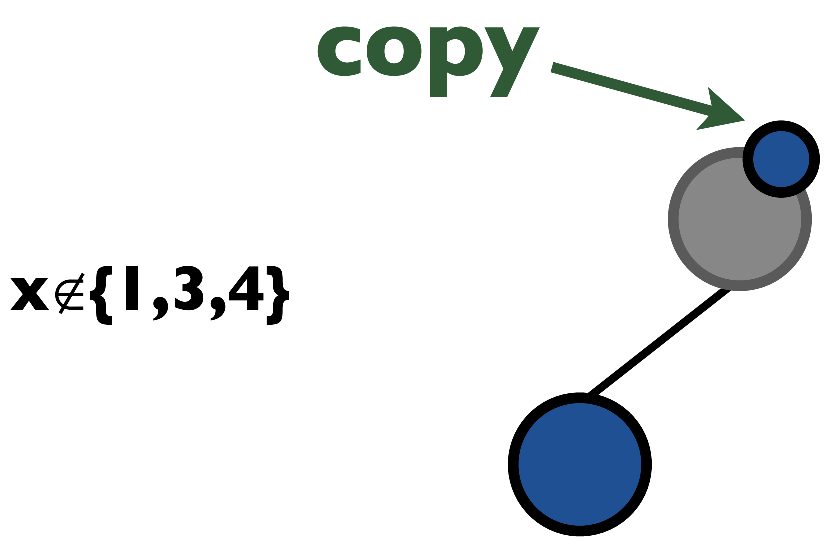
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



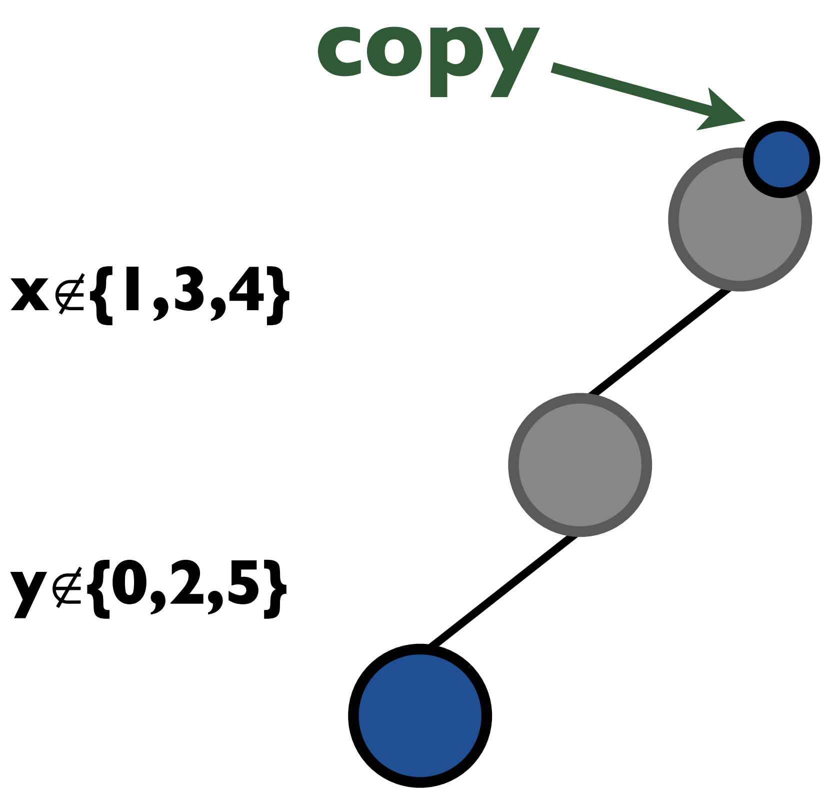
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



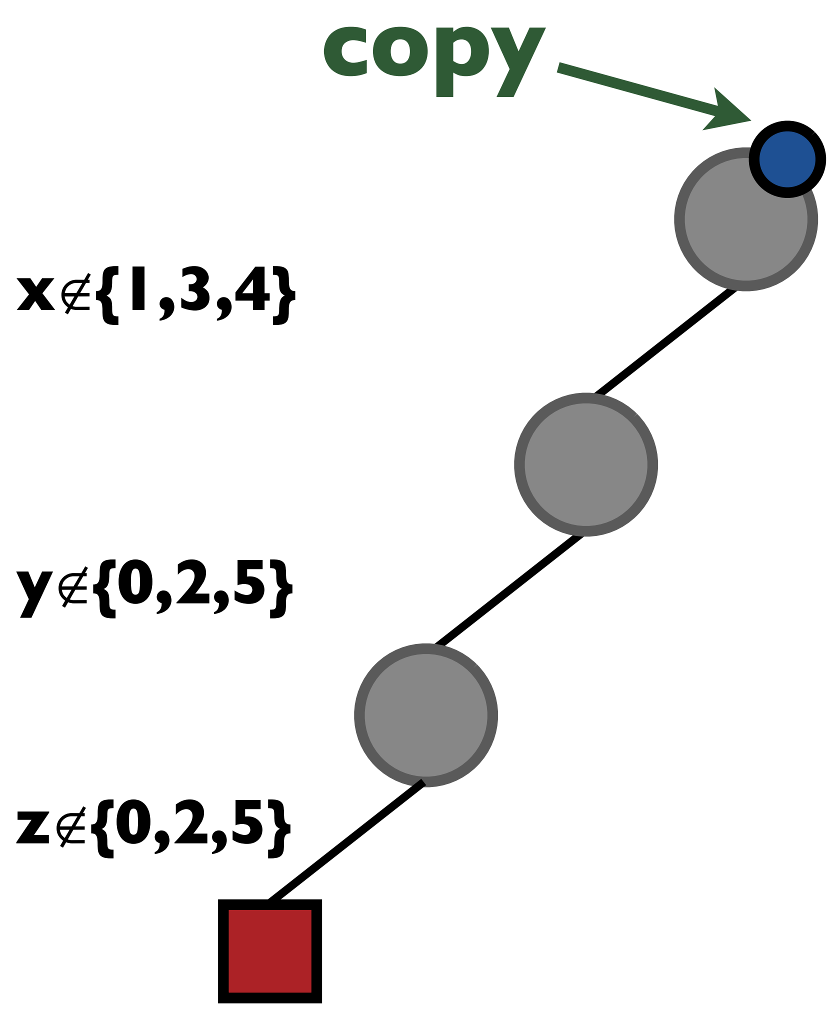
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



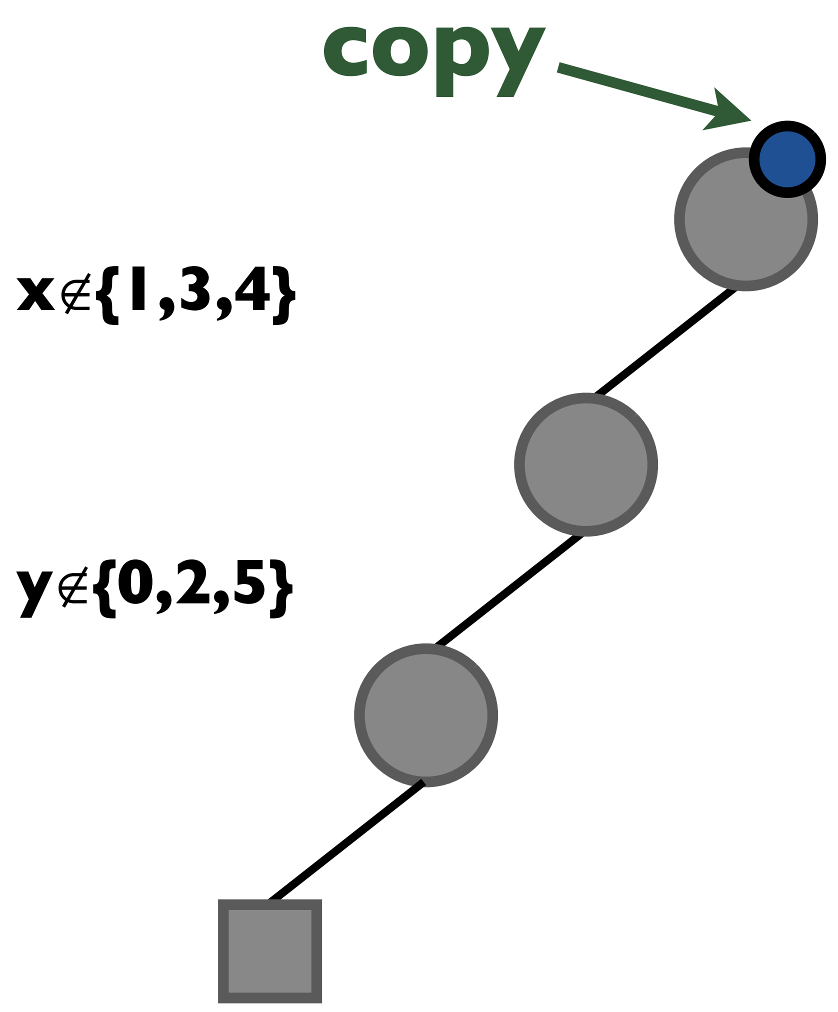
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



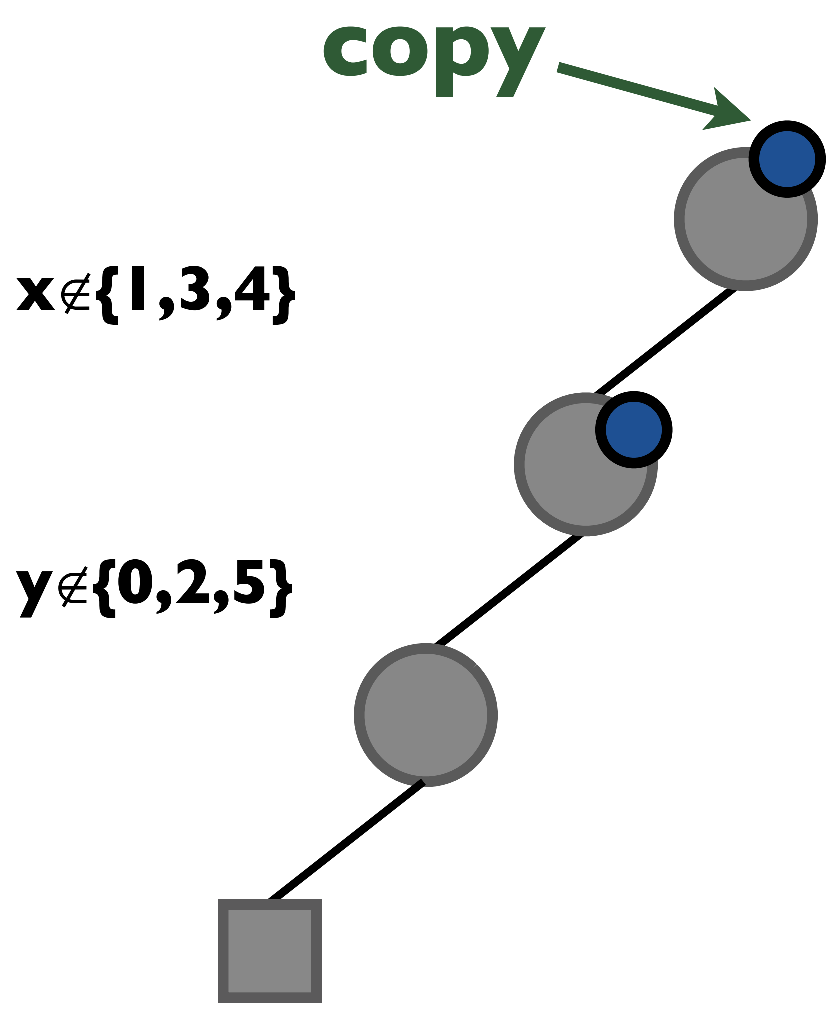
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



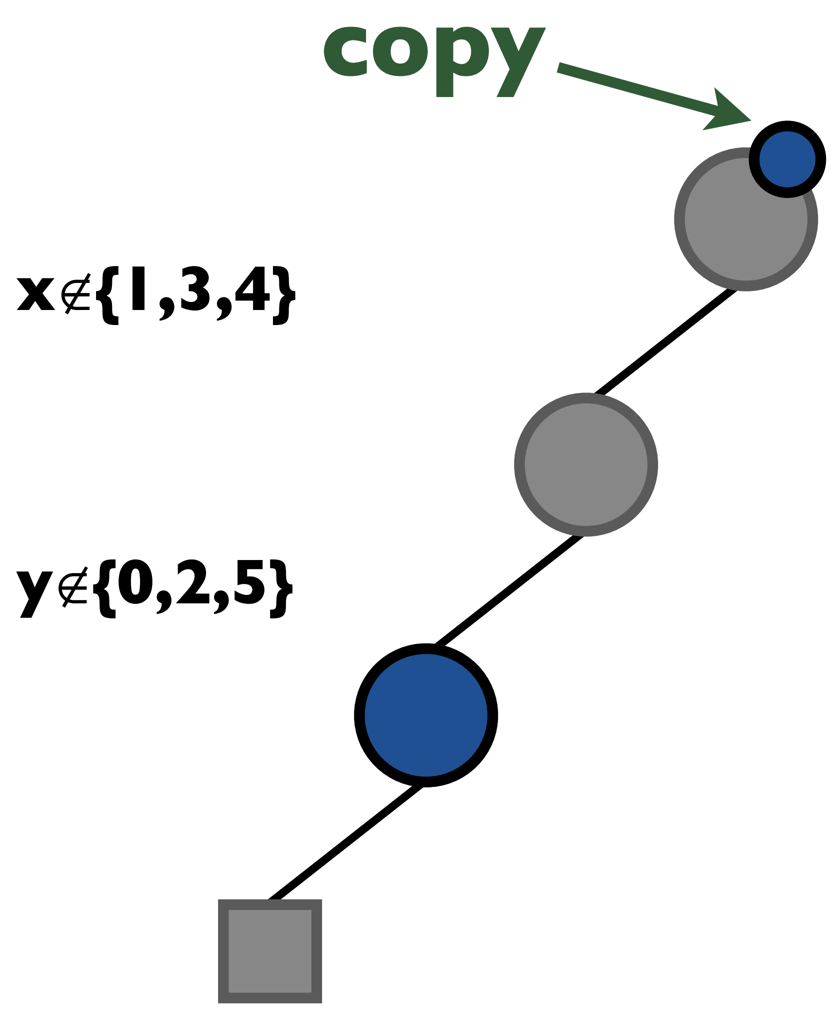
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



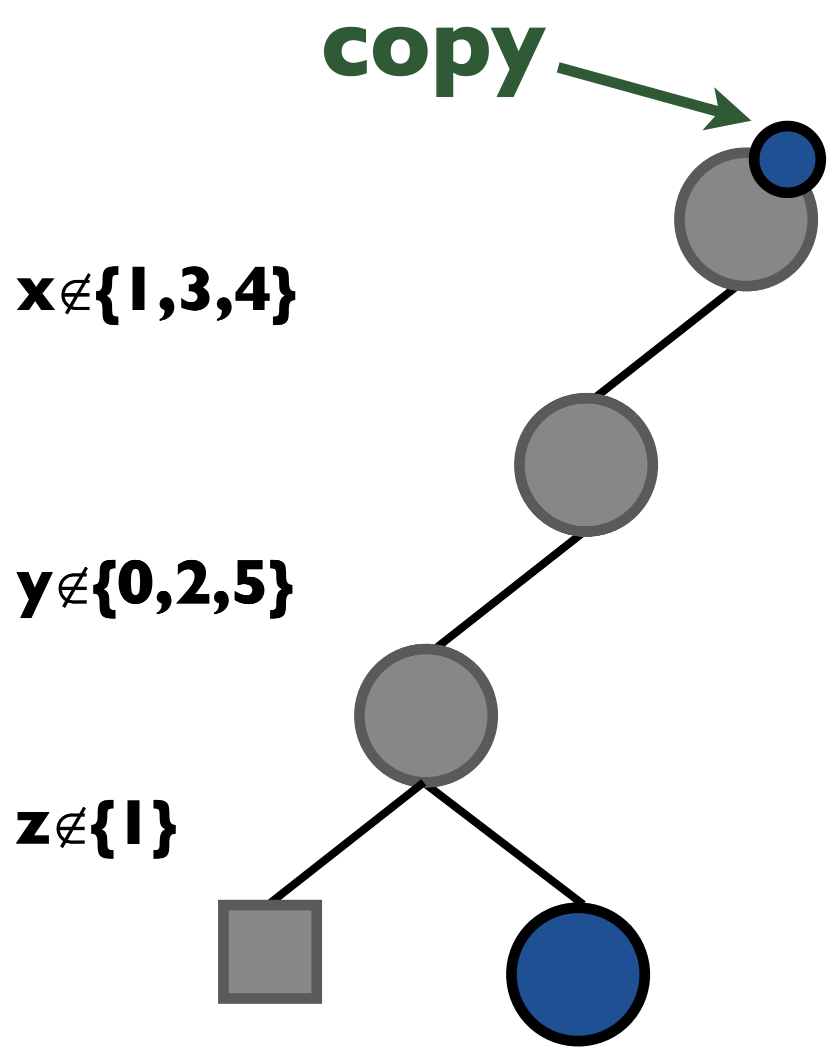
Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



Local State

- Independent state per node
- Maintained by **copying** and **recomputation**



Recomputing Propagator State

- Propagator state can only depend on domains
- **Naive** approach: always recompute
  - no more incrementality

Recomputing Propagator State

- Propagator state can only depend on domains
- **Naive** approach: always recompute
  - no more incrementality
- **Better:** recompute only after backtracking
  - still have "forwards incrementality"
  - far less recomputation
  - needs neither trailing nor copying

Pros & Cons: Global State

Pros & Cons: Global State

+ Only invest (trail) for actual changes

Pros & Cons: Global State

+ Only invest (trail) for actual changes
+ Cheap backtracking if little changes

Pros & Cons: Global State

+ Only invest (trail) for actual changes
+ Cheap backtracking if little changes
+ Easy to share information between nodes (e.g. objective value)

Pros & Cons: Global State

- + Only invest (trail) for actual changes
- + Cheap backtracking if little changes
- + Easy to share information between nodes (e.g. objective value)
- + Watched literals

Pros & Cons: Global State

+ Only invest (trail) for actual changes
+ Cheap backtracking if little changes
+ Easy to share information between nodes (e.g. objective value)
+ Watched literals
- Strictly DFS

Pros & Cons: Global State

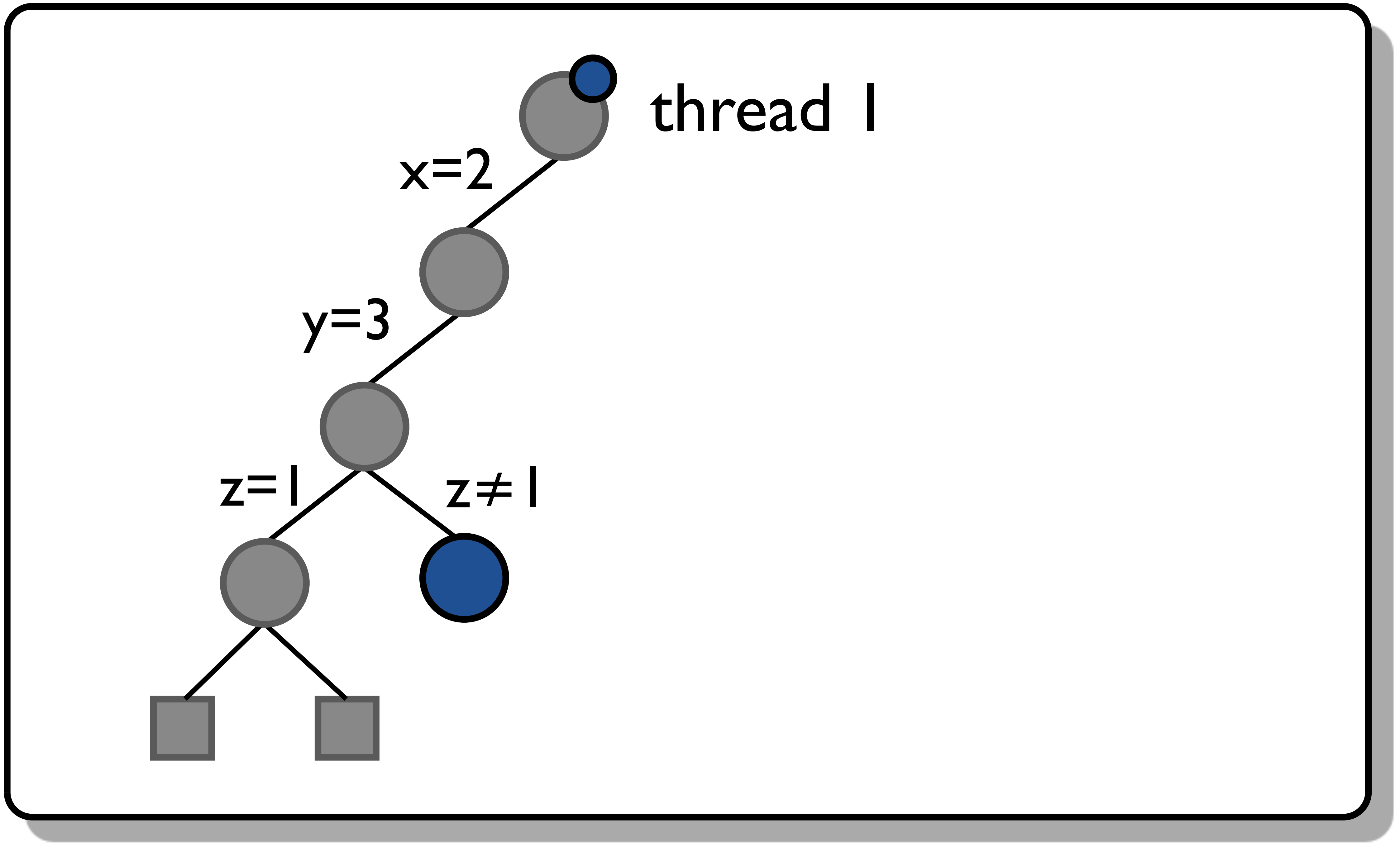
+ Only invest (trail) for actual changes
+ Cheap backtracking if little changes
+ Easy to share information between nodes (e.g. objective value)
+ Watched literals
- Strictly DFS
- Global state incompatible with shared-memory parallelism

Pros & Cons: Local State

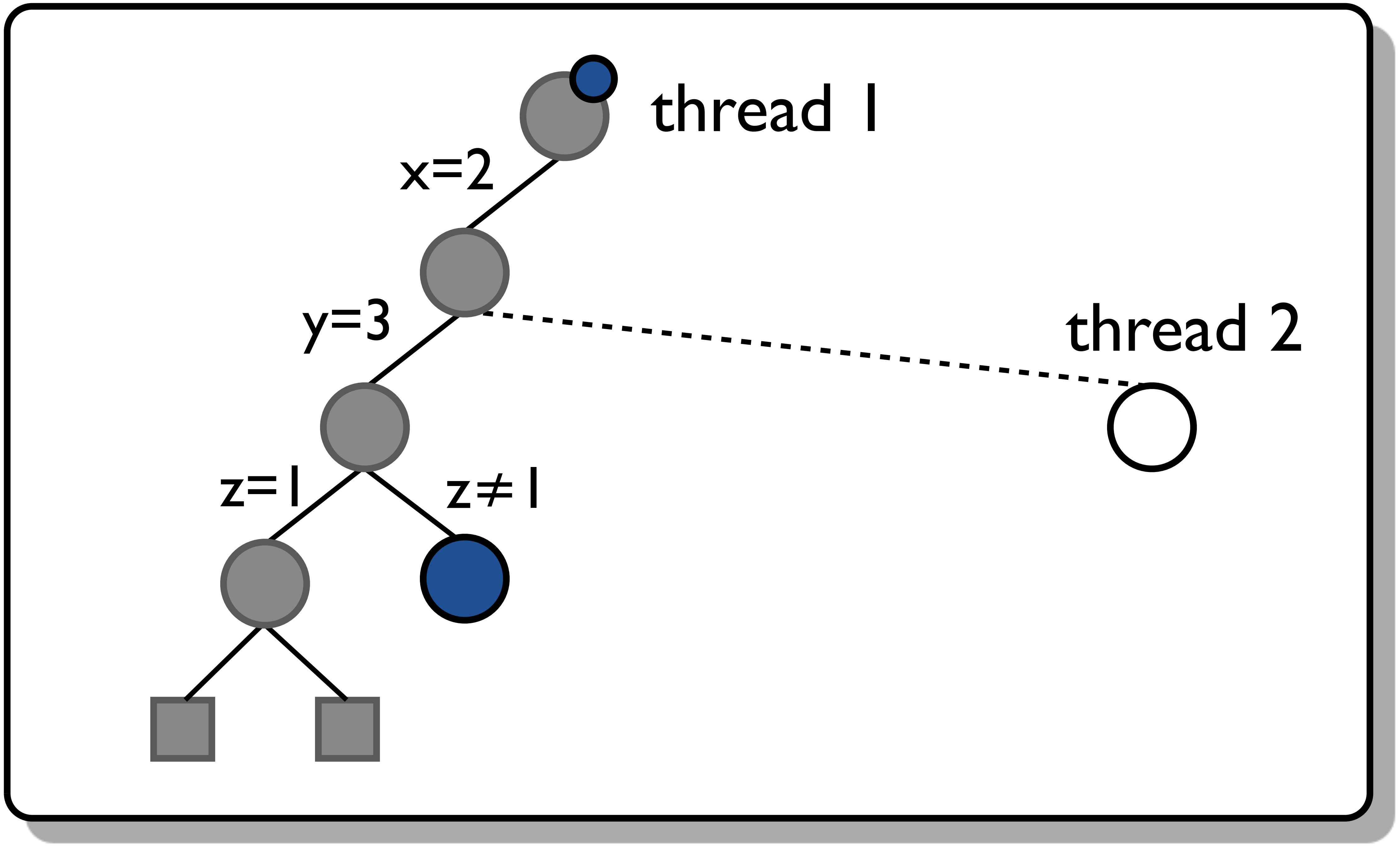
Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nodes
  - parallel: work stealing through recomputation

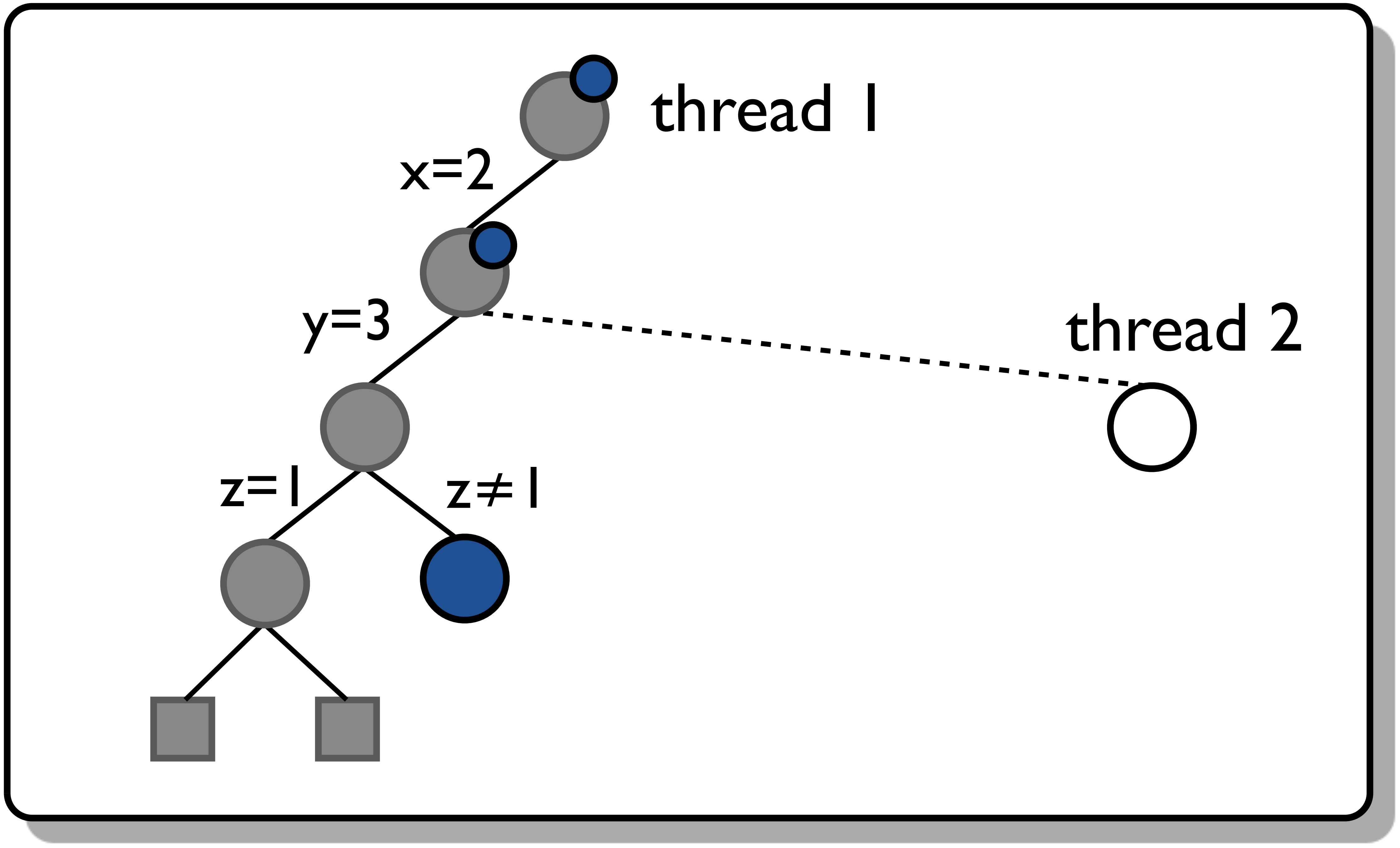
Pros & Cons: Local State



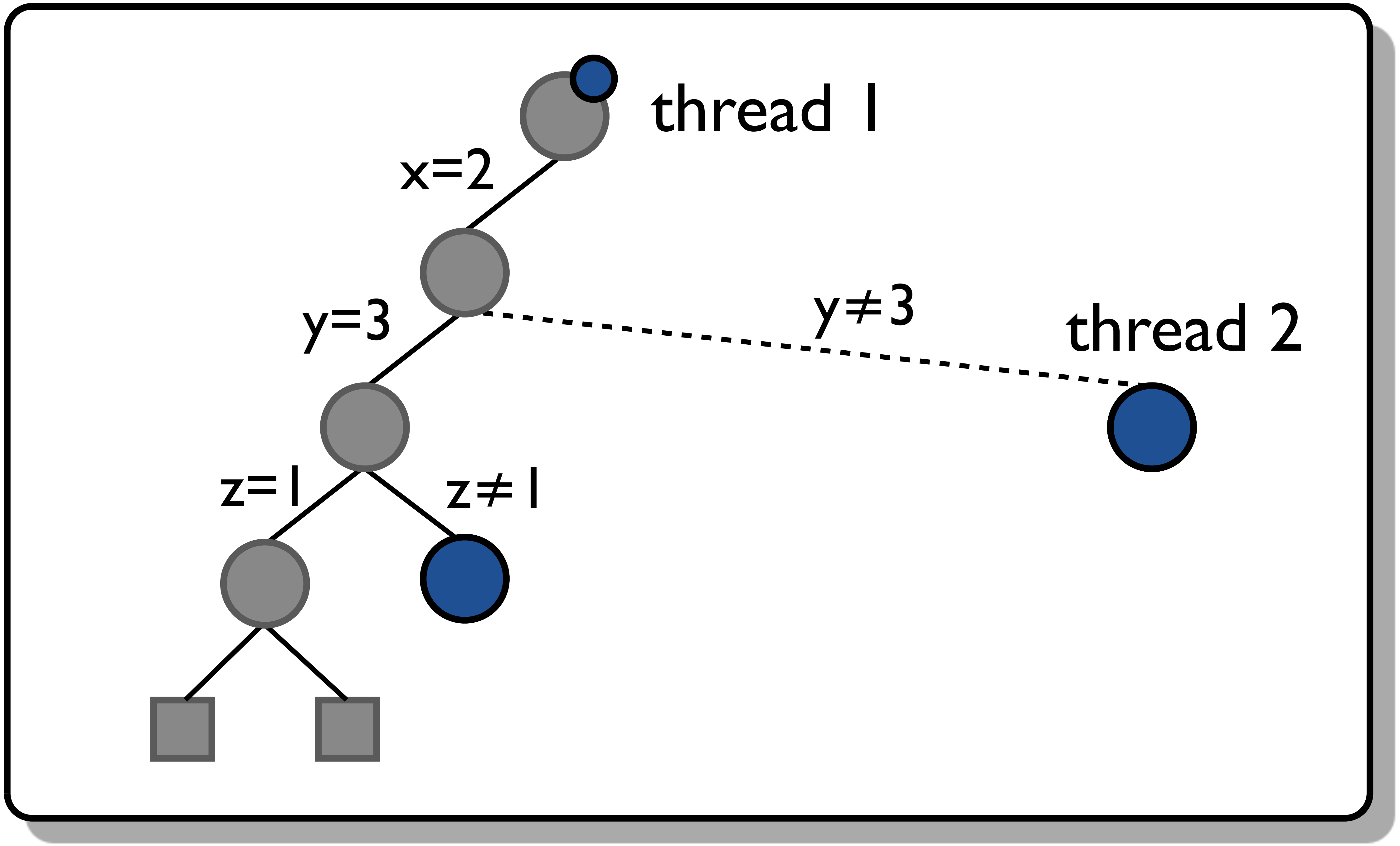
Pros & Cons: Local State



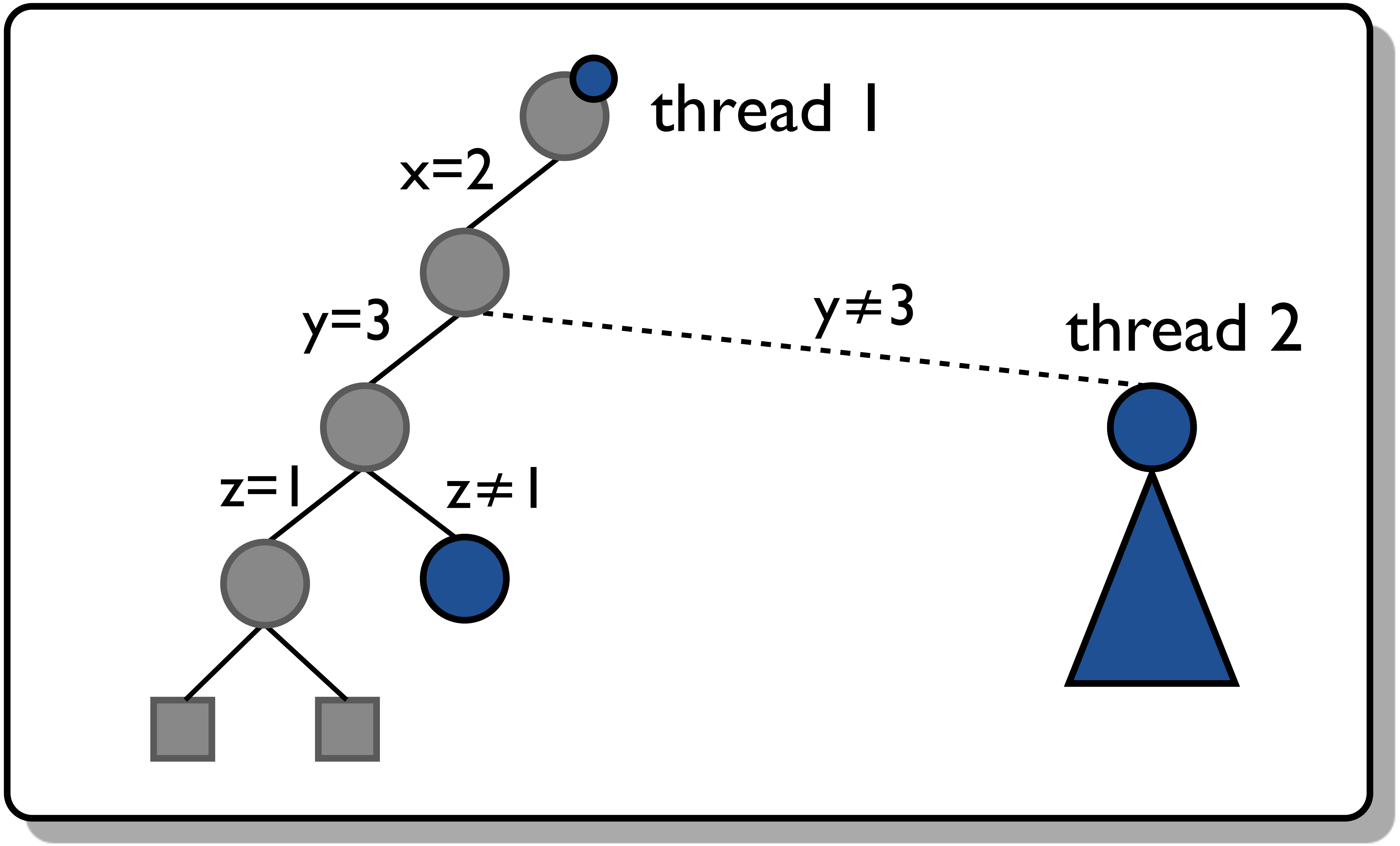
Pros & Cons: Local State



Pros & Cons: Local State



Pros & Cons: Local State



Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nodes
  - parallel: work stealing through recomputation

Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nodes
  - parallel: work stealing through recomputation

  Confidence-based Work Stealing
  don't miss it this afternoon!

Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nodes
  - parallel: work stealing through recomputation

    Confidence-based Work Stealing
    don't miss it this afternoon!

+ Control memory consumption

Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nodes
  - parallel: work stealing through recomputation

    Confidence-based Work Stealing
    don't miss it this afternoon!
+ Control memory consumption
- Copying may be costly if little changes

Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nodes
  - parallel: work stealing through recomputation

    Confidence-based Work Stealing
    don't miss it this afternoon!
+ Control memory consumption

– Copying may be costly if little changes

– Backtracking may require more work

Pros & Cons: Local State

+ Greatly simplifies involved search strategies:
  - A^*: jump between "open" nod
  - parallel: work stealing throug

    Confidence-based Work
    don't miss it this afternoon!
+ Control memory consumption
- Copying may be costly if little changes
- Backtracking may require more work

architecture

Pros & Cons: Local State

- + Greatly simplifies involved search strategies:
  - A*: jump between "open" nod
  - parallel: work stealing throug

    Confidence-based Work
    don't miss it this afternoon!
- + Control memory consumptio
- − Copying may be costly if little
- − Backtracking may require mor

architecture

efficiency

A Hybrid System

- **Best of both worlds**
- Anything trailed can also be copied / recomputed

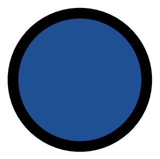
architecture

- **Realistic evaluation**
- Same objects in trailed and copied / recomputed versions

efficiency

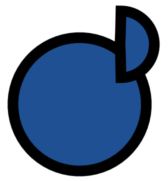
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



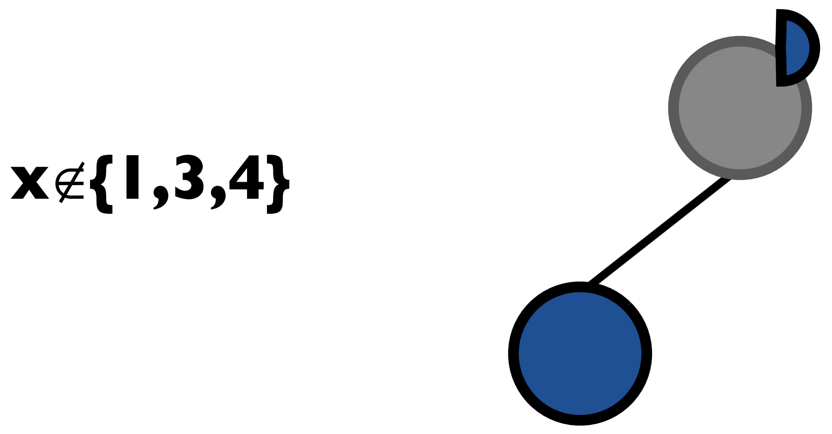
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute

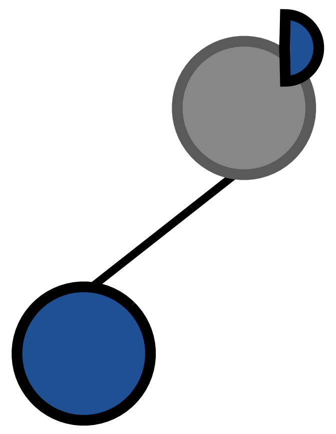


Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute

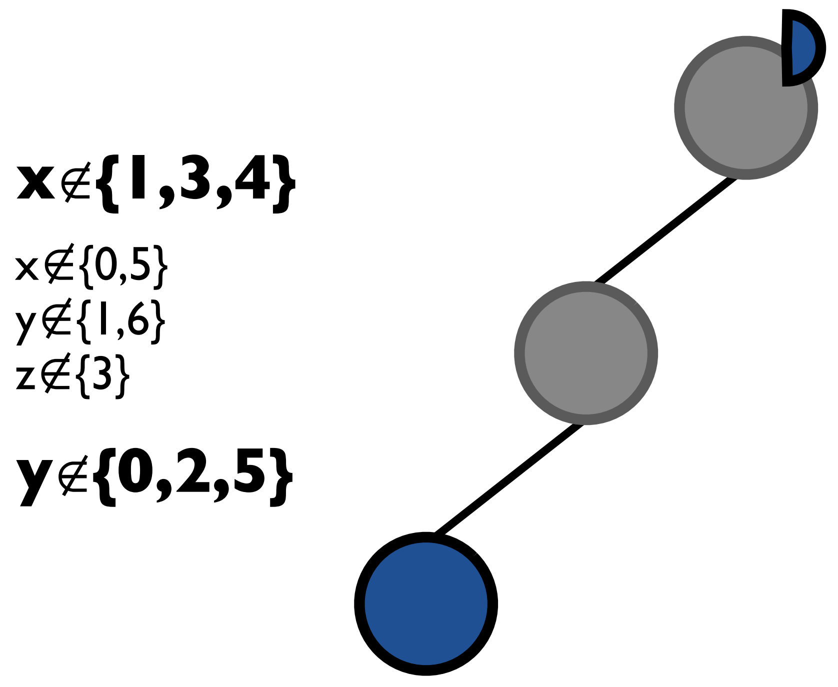
x∉{1,3,4}

x∉{0,5}
y∉{1,6}
z∉{3}



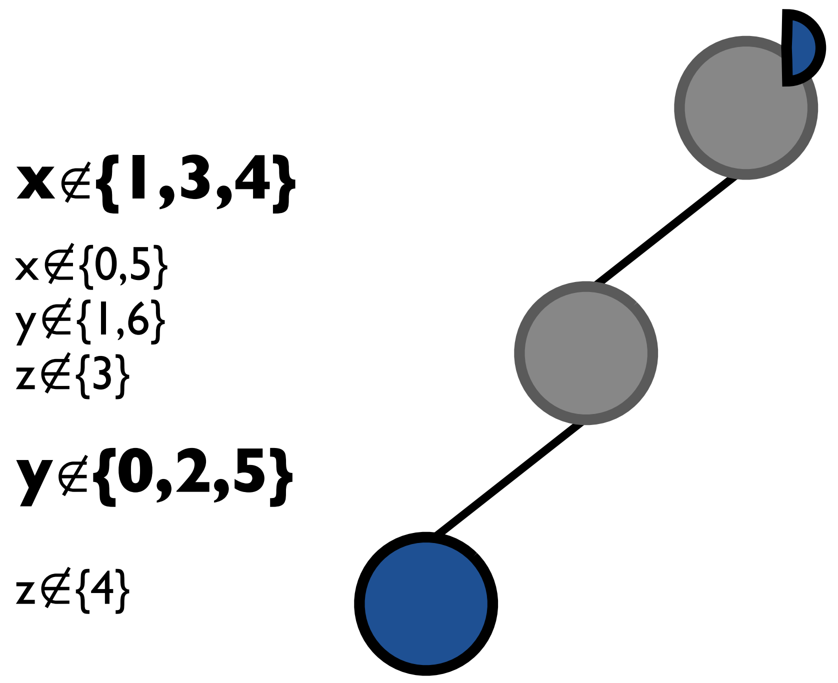
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



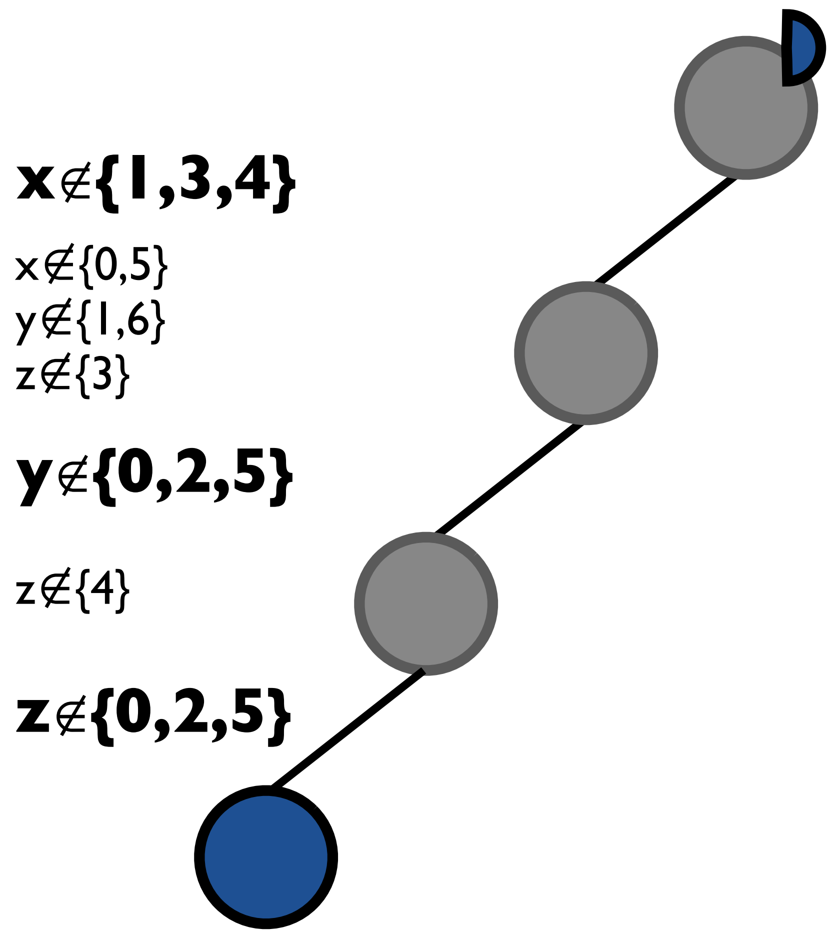
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



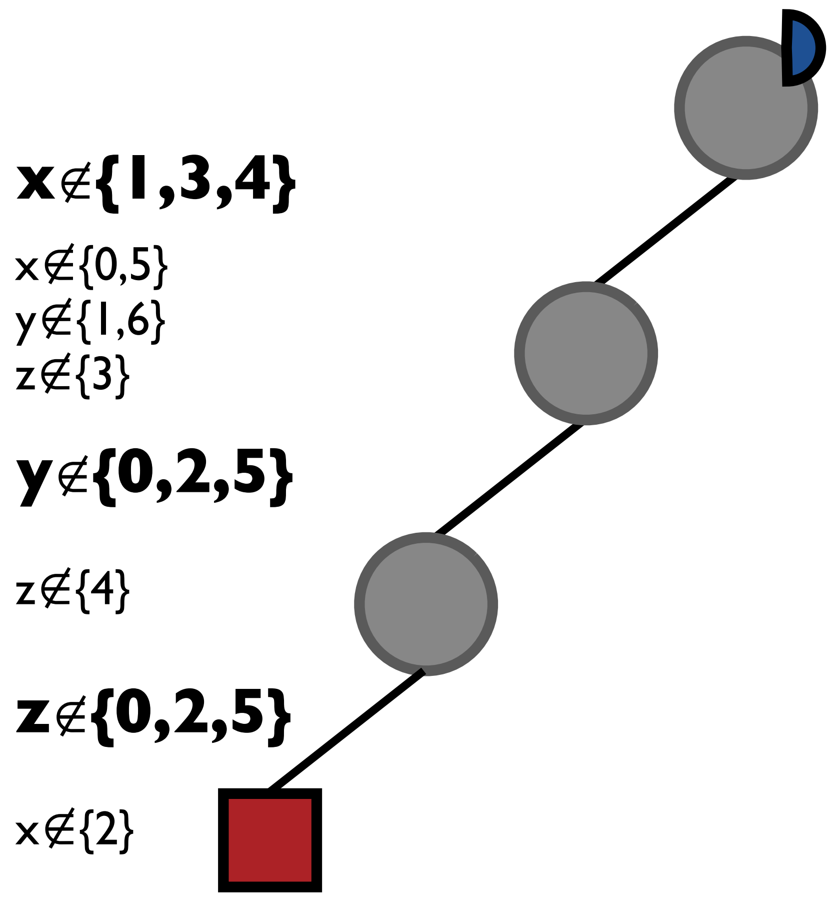
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



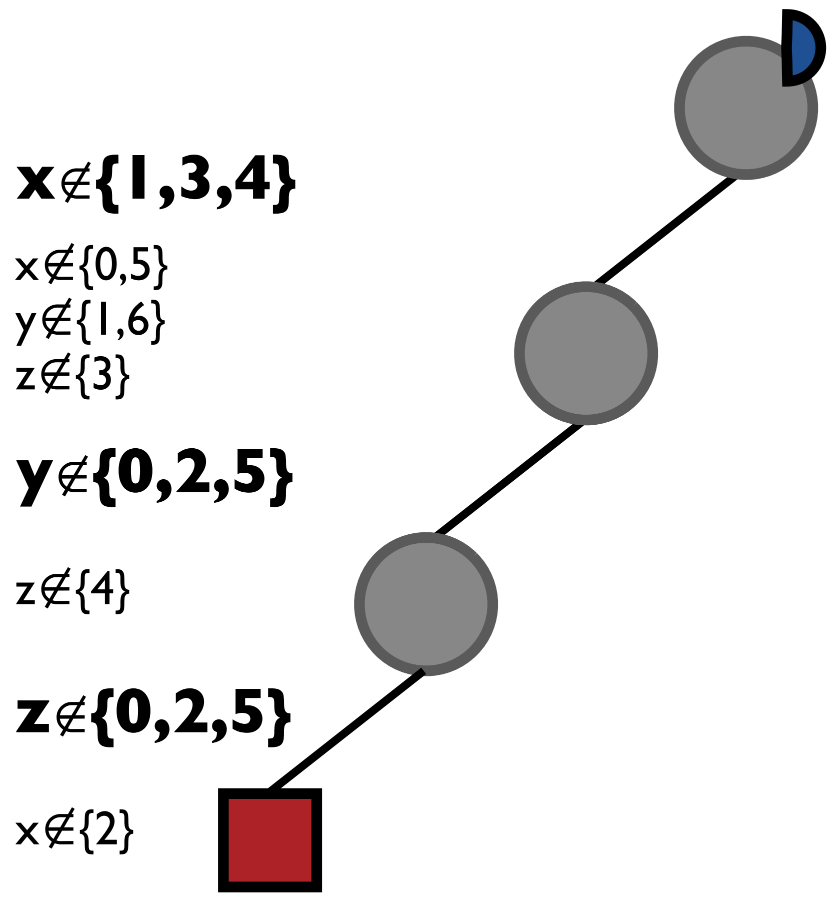
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



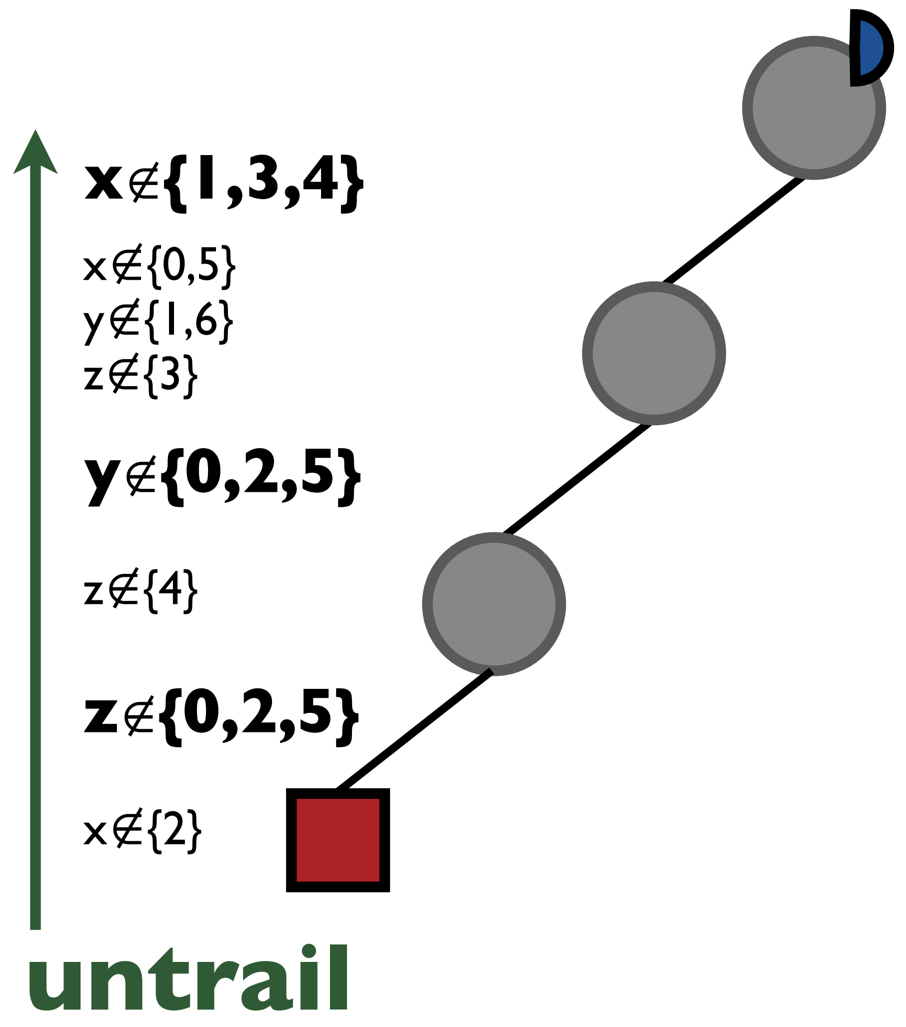
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



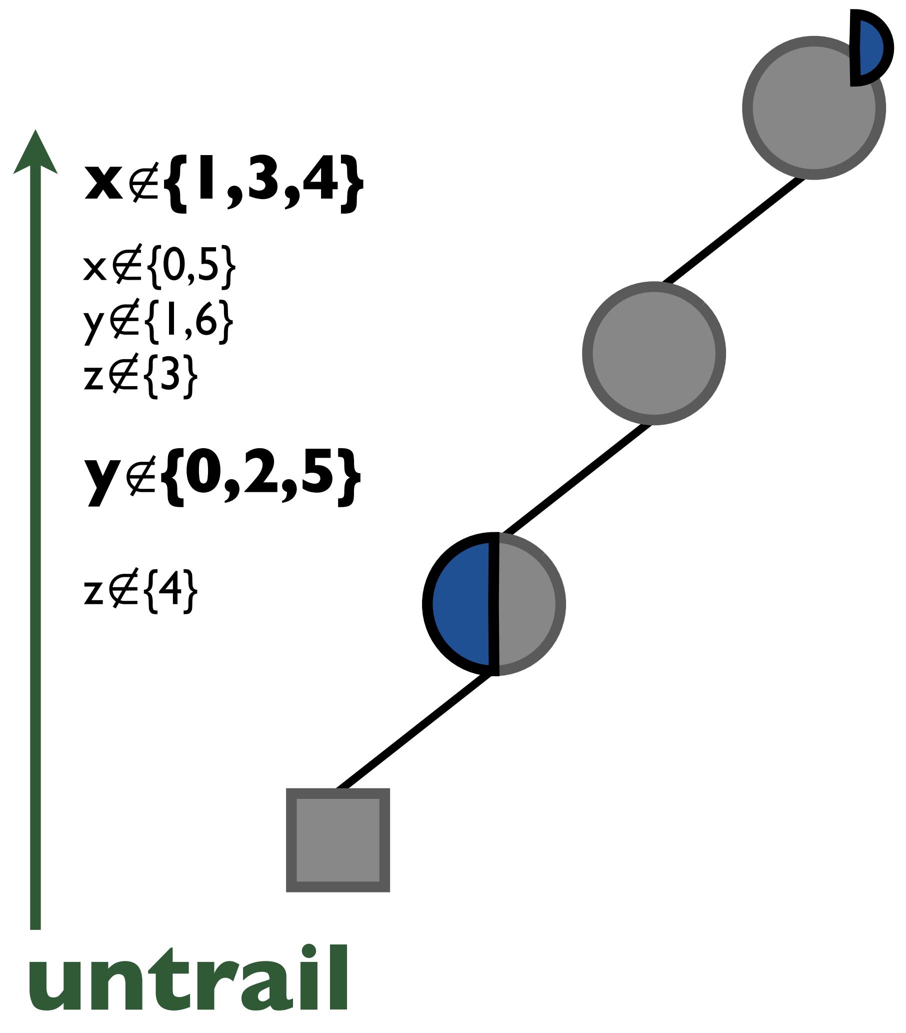
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



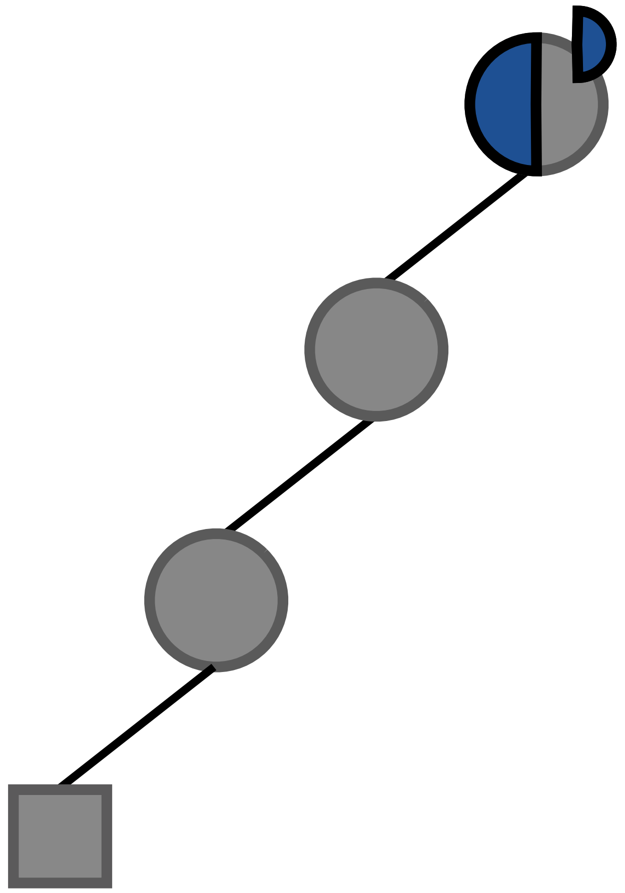
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



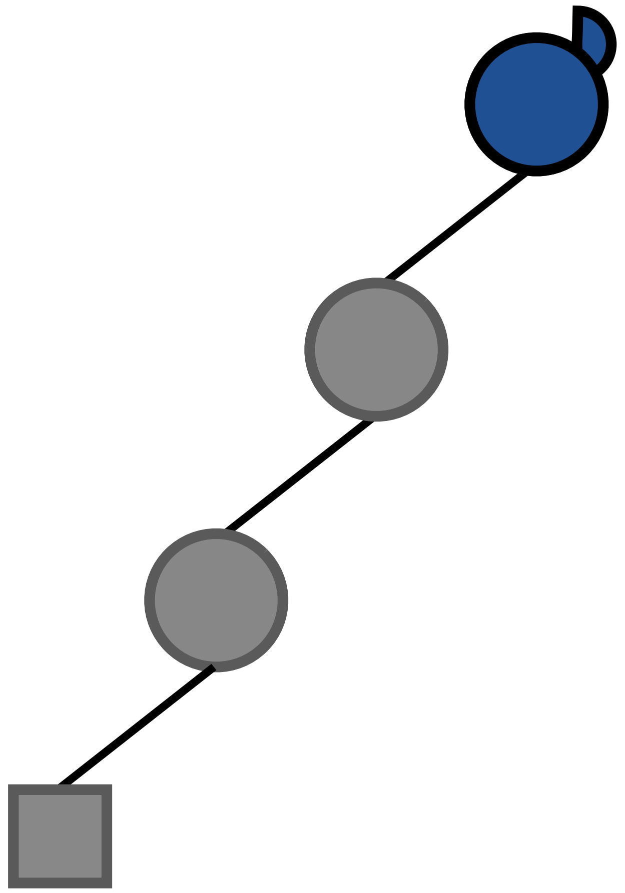
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



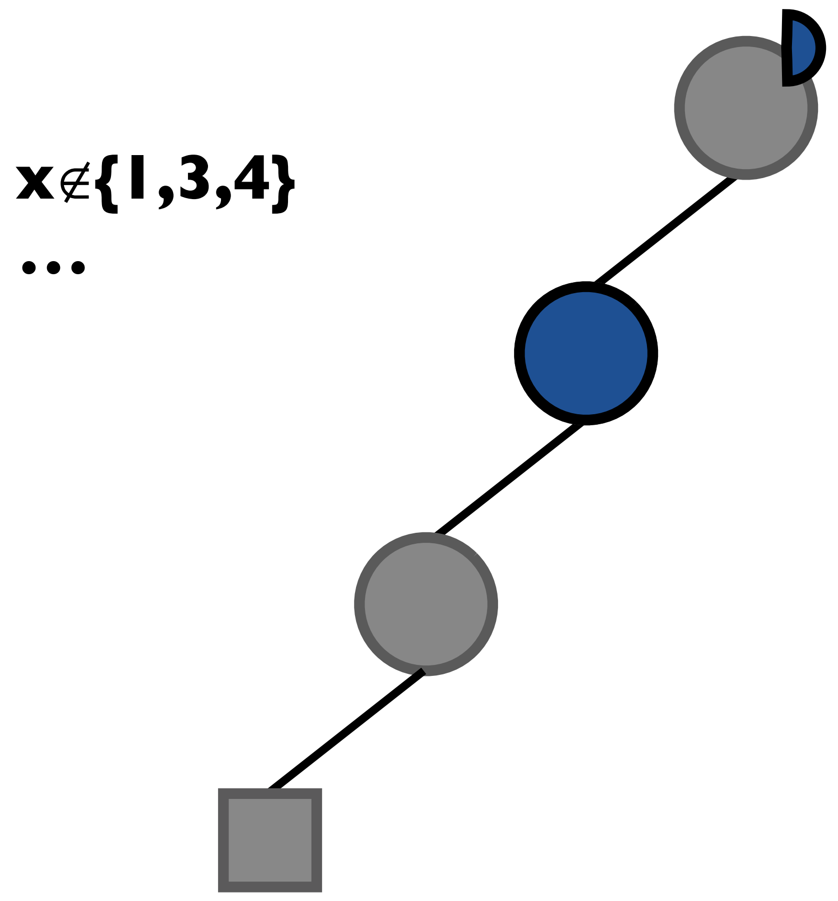
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



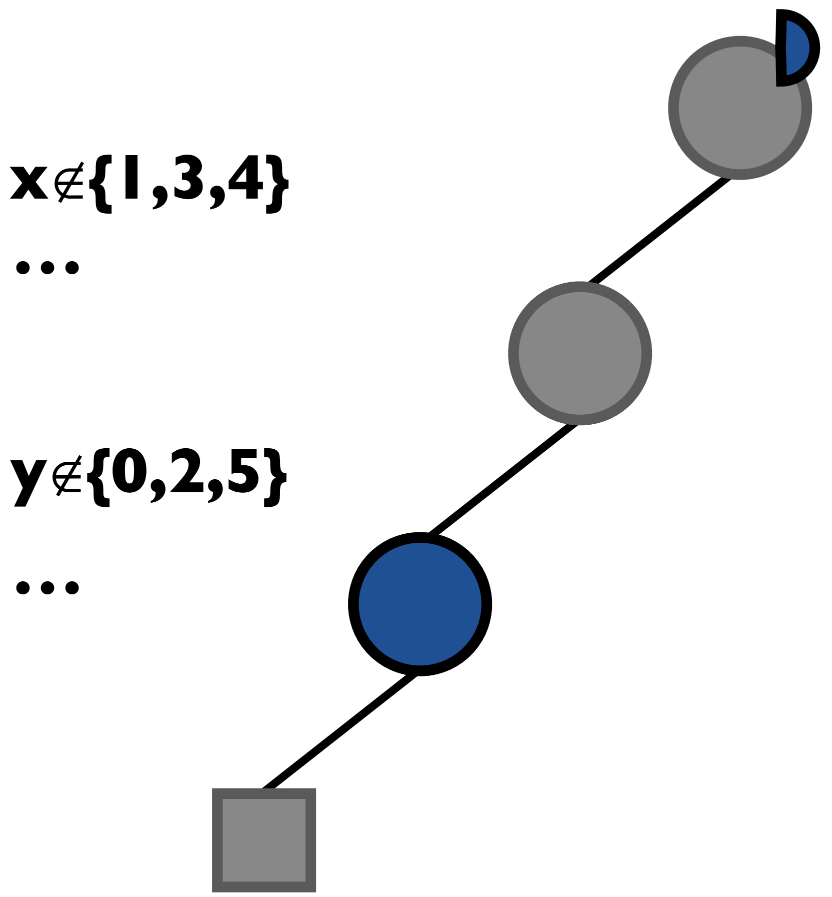
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



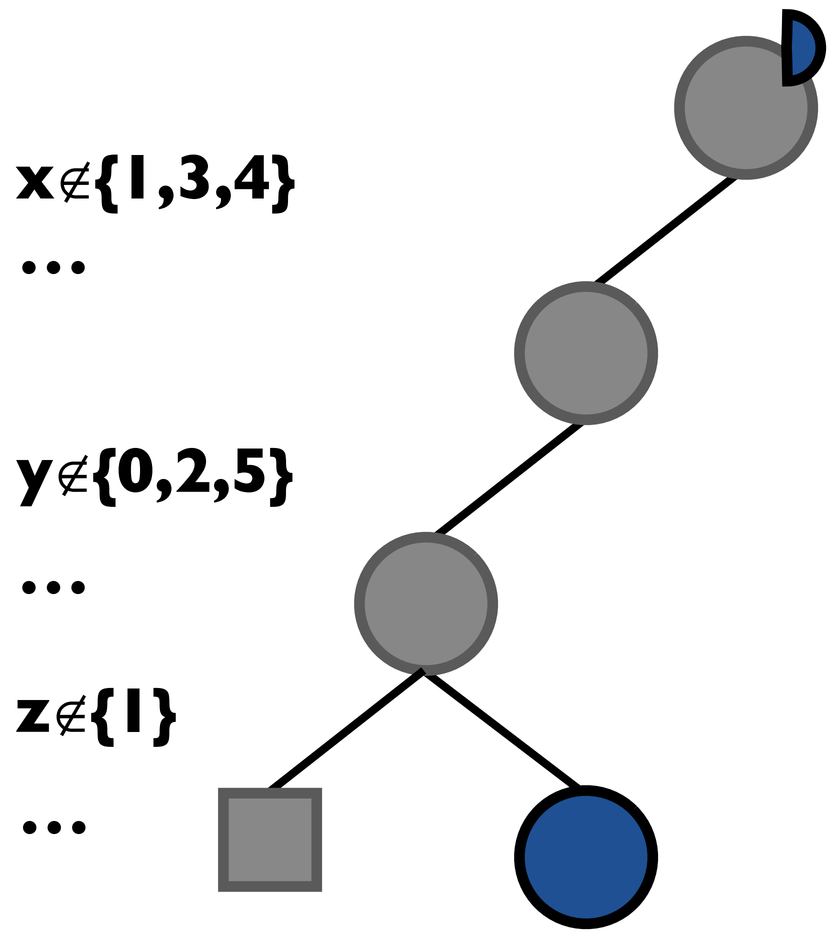
Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



Hybrid State Restoration

- Copy part of the state
- Trail other part of the state
- On backtracking, untrail to closest copy above common ancestor, then recompute



A Hybrid System

- Based on **Gecode**
  - base system uses recomputation
  - added global trail
  - added propagators with trailed and backtrack-safe state
  - added trailed integer/Boolean variable domains and dependencies
- First completely hybrid, state-of-the-art solver

Evaluation

Related Work

- Simulation of trailing in Mozart [Schulte 1999]
  - no runtime evaluation
  - memory performance just an estimate
- Integration of *coarse-grained trailing* and recomputation in Figaro [Choi et al. 2001]
  - prototype system, non-competitive runtime

Evaluation Scenarios

- Local:

  local domains & propagators (standard Gecode)

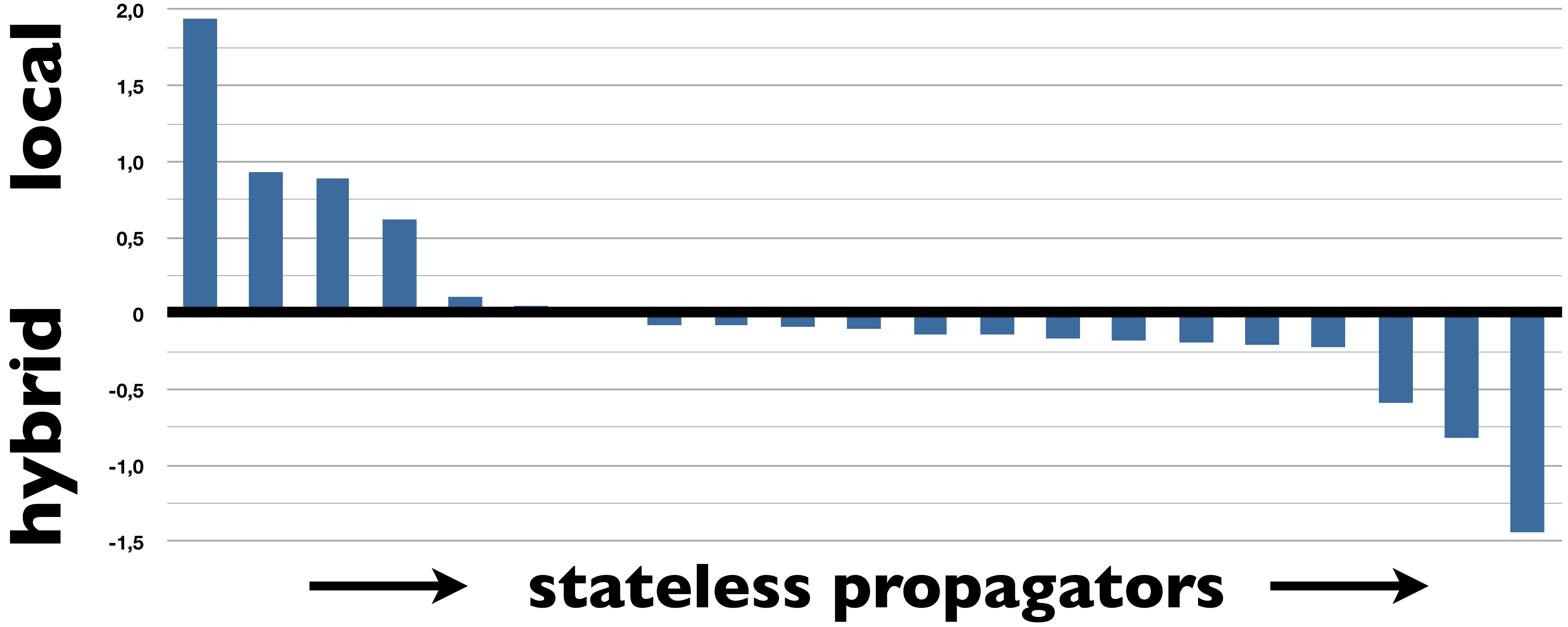
- Hybrid:

  local domains, global propagators

- Global:

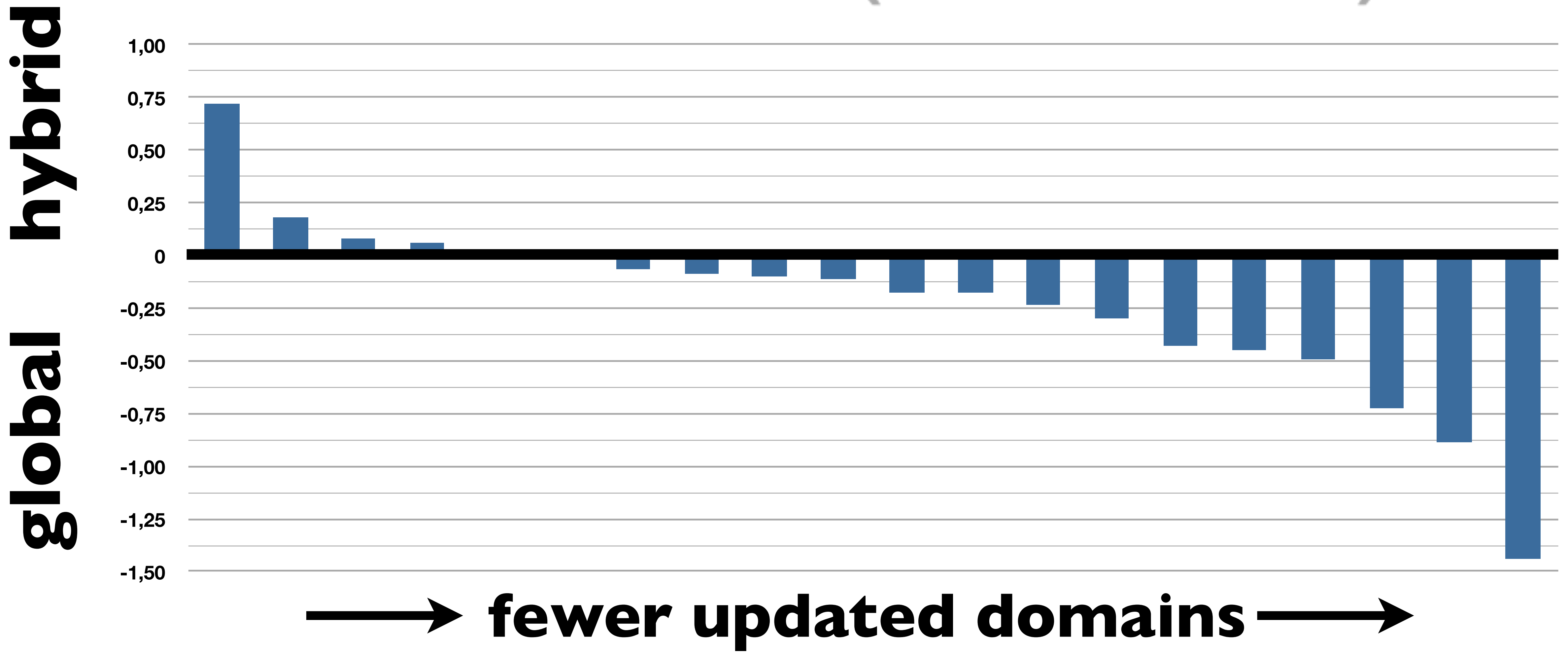
  global domains & propagators

Propagators (runtime)



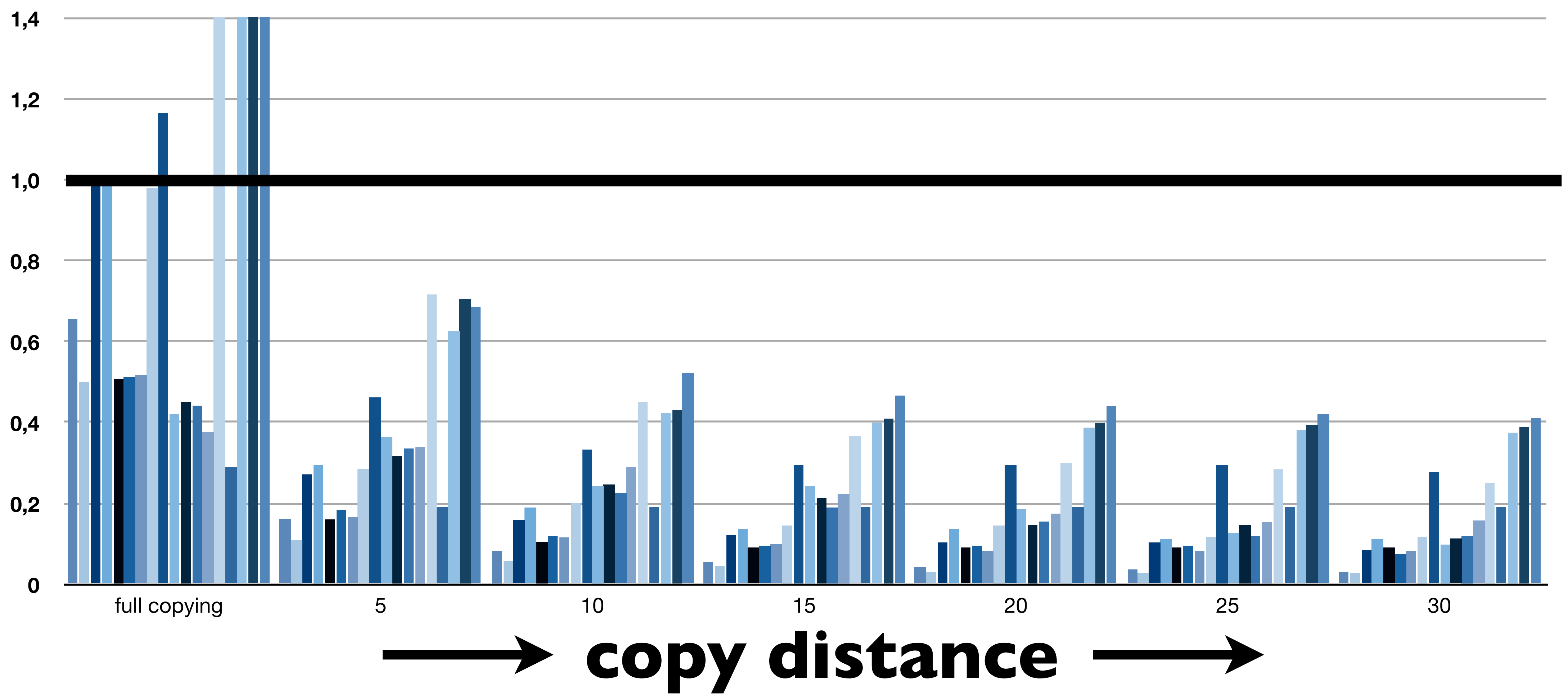
- At most factor 3 apart
- Propagators without state should not be copied

Domains (runtime)



- At most factor 2.4 apart, usually less (except one example, not shown, factor 24)
- Most influential: percentage of updated domains

Memory



- Recomputation uses less memory than trailing
- On average 20% memory at distance 10

Summary

- Each strategy is best for some examples
- Trailing is more robust w.r.t. runtime
- Recomputation is more robust w.r.t. memory

- Future work:

  parallel search using hybrid approach

Summary

- Each strategy is best for some examples
- Trailing is more robust w.r.t. runtime
- Recomputation is more robust w.r.t. memory

- Future work:

  parallel search using hybrid approach

Thanks!