

Component-based design in Tako (a case study)

Arun Sudhir • Gregory Kulczycki • Jyotindra Vasudeo

Gregory Kulczycki

Gregory Kulczycki

Arun Sudhir

Gregory Kulczycki

Arun Sudhir

Jyotindra Vasudeo





Falls Church, VA

Component-based design in Tako **(a case study)**

(1) What is Tako?

(2) What is Tako design?

(1) What is Tako?

Tako \approx Java + Resolve



What is Resolve?

**integrated
programming & specification
language**

full formal verification

(full = heavyweight)

Statically prove...

CODE is correct w.r.t. **SPEC**

Verifying Compiler Grand Challenge

Tony Hoare, 2003

Resolve and language design

Make things as simple as possible...



but no simpler.

– Albert Einstein

Simplicity in Reasoning

The diagram consists of two circular nodes. The left node is purple and contains the text 'Simplicity in Reasoning'. The right node is blue and contains the text 'Sophisticated Language Features'. Two thick, curved purple arrows connect the nodes: one points from the blue node to the purple node, and the other points from the purple node to the blue node, indicating a reciprocal relationship.

Sophisticated
Language
Features

!!!



no pointers

no inheritance

no concurrency



pointers

inheritance

concurrency

(but disciplined)

Pointers

Pointers = References

⇒ Aliasing

{ x = true }

y := false

{ x = ??? }

{ x = true }

y := false

{ x = ??? }

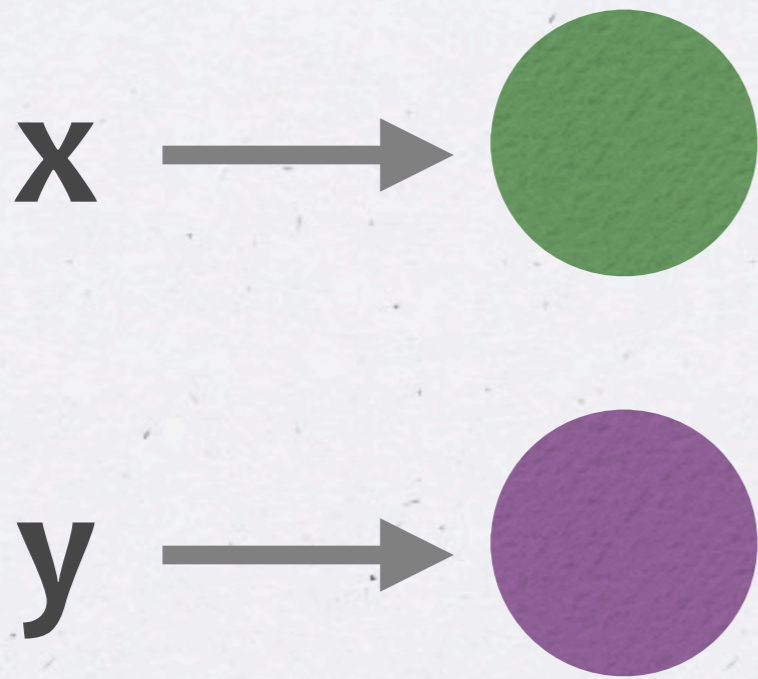
**Is x
aliased
to y?**



reference copying \Rightarrow aliasing

object copying \Rightarrow expensive

$x ::= y$



before



after

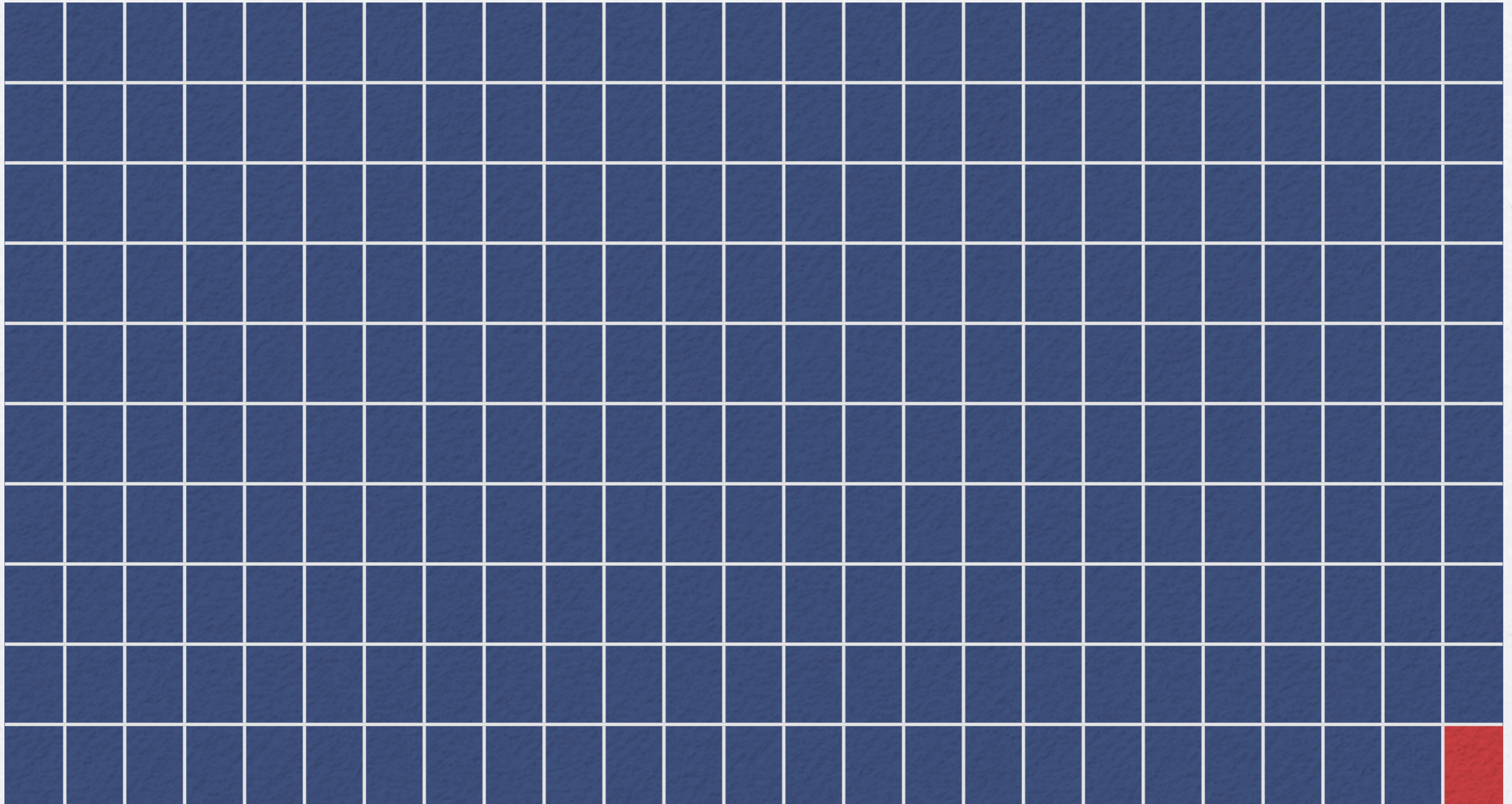
Using Resolve/C++

Joe Hollingsworth et al, 2000

100,000

lines of code

swapping/copying-based



pointer-based



What is Resolve?

(1) Goal – full verification

(2) Language – swapping

(1) What is Tako?

Tako \approx Java + Resolve



sex-appeal



sex-appeal
popular



sex-appeal
popular
not logical



sex-appeal
popular
not logical



idealist

sex-appeal
popular
not logical



idealist
rational

sex-appeal
popular
not logical



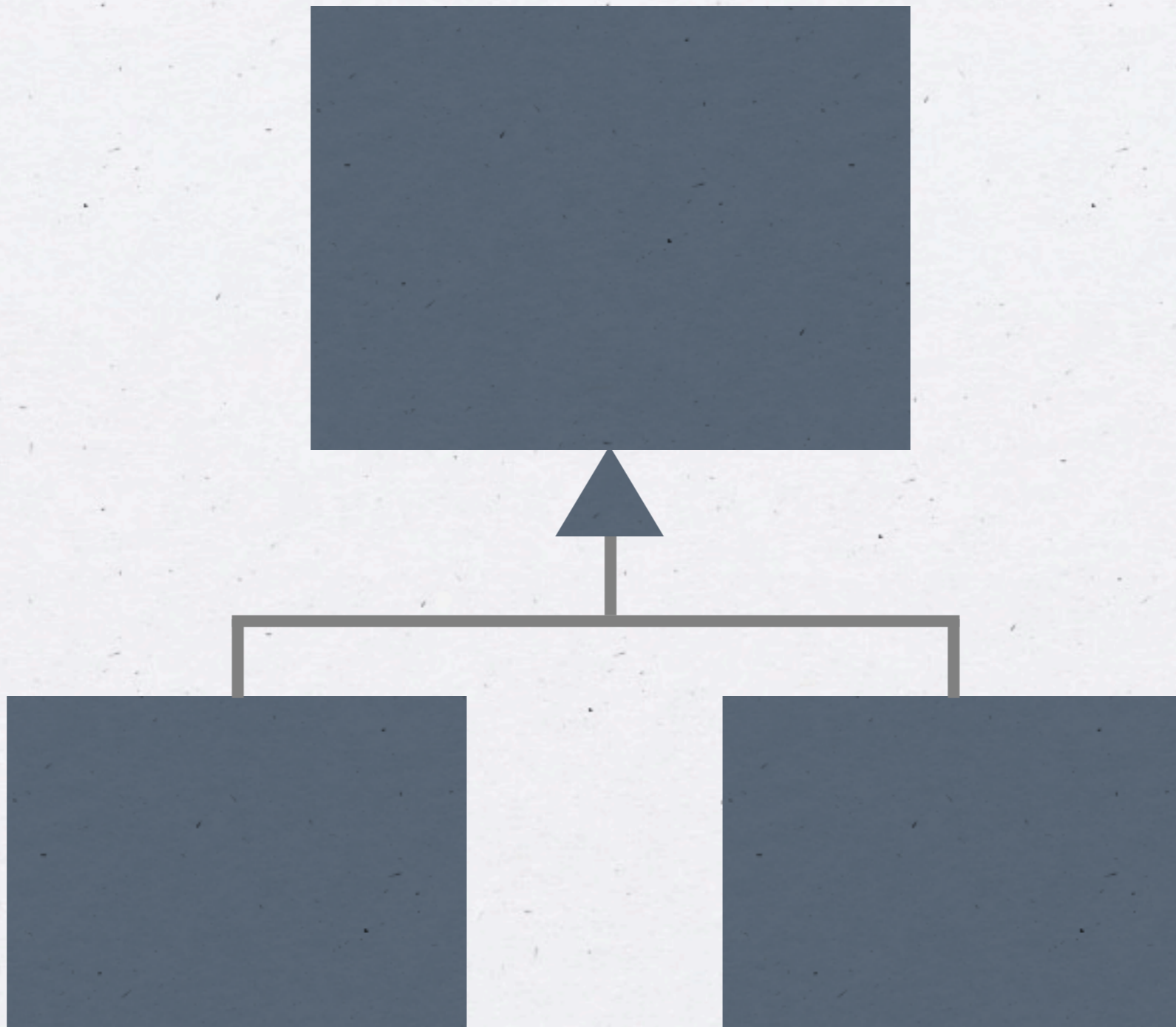
idealist
rational
know-it-all

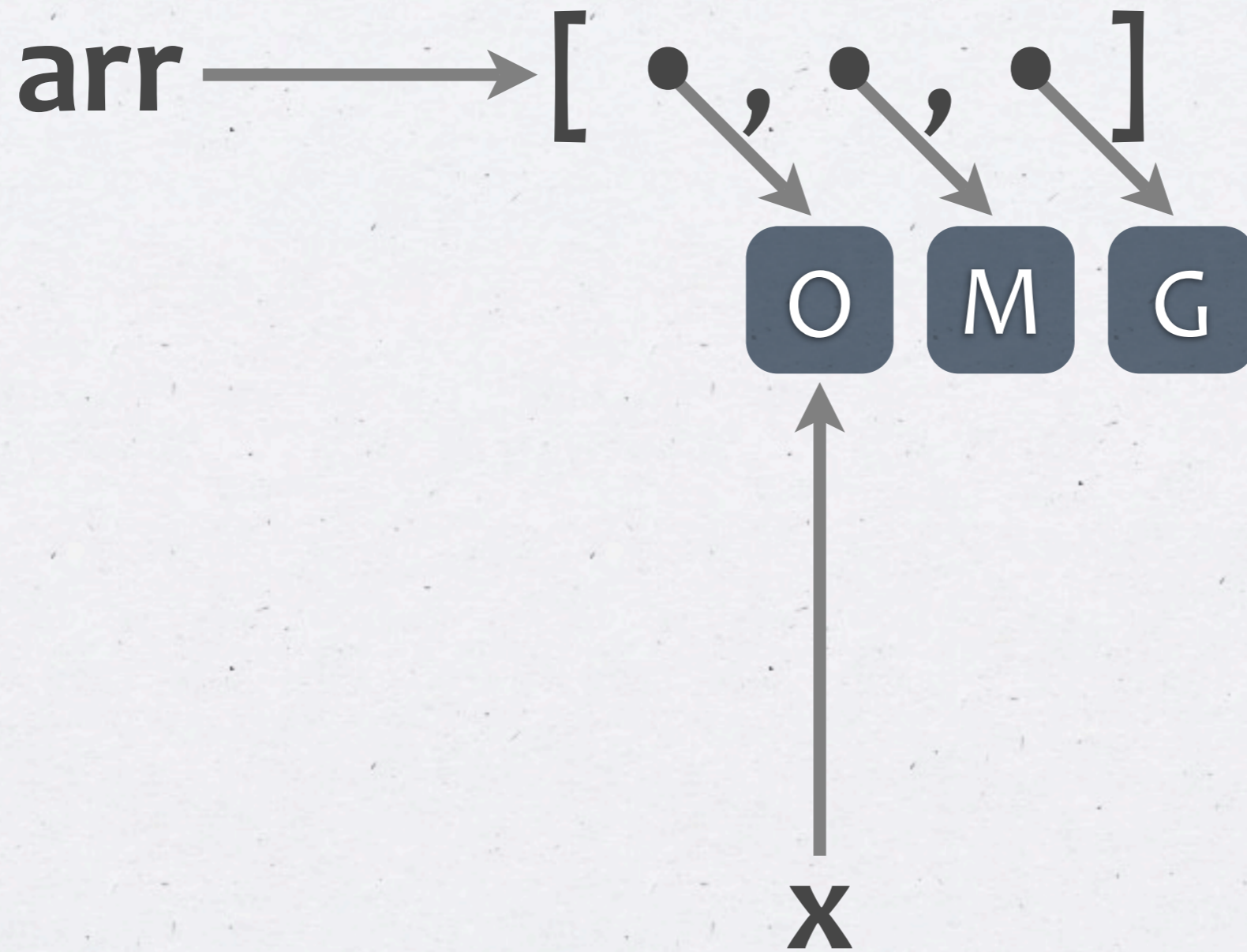
sex-appeal
popular
not logical



idealist
rational
know-it-all

Tako is a compromise





Why Tako?

(1) Teach formal reasoning

(2) Simplify informal reasoning

Pre-state:

$p = \langle \Psi, \Phi \rangle$

$q = \langle \Psi, \Delta, \Psi, \Delta \rangle$

$t = \Delta$

$p := q;$

$q.enqueue(t);$

$t.clear();$

// initialize t to Φ

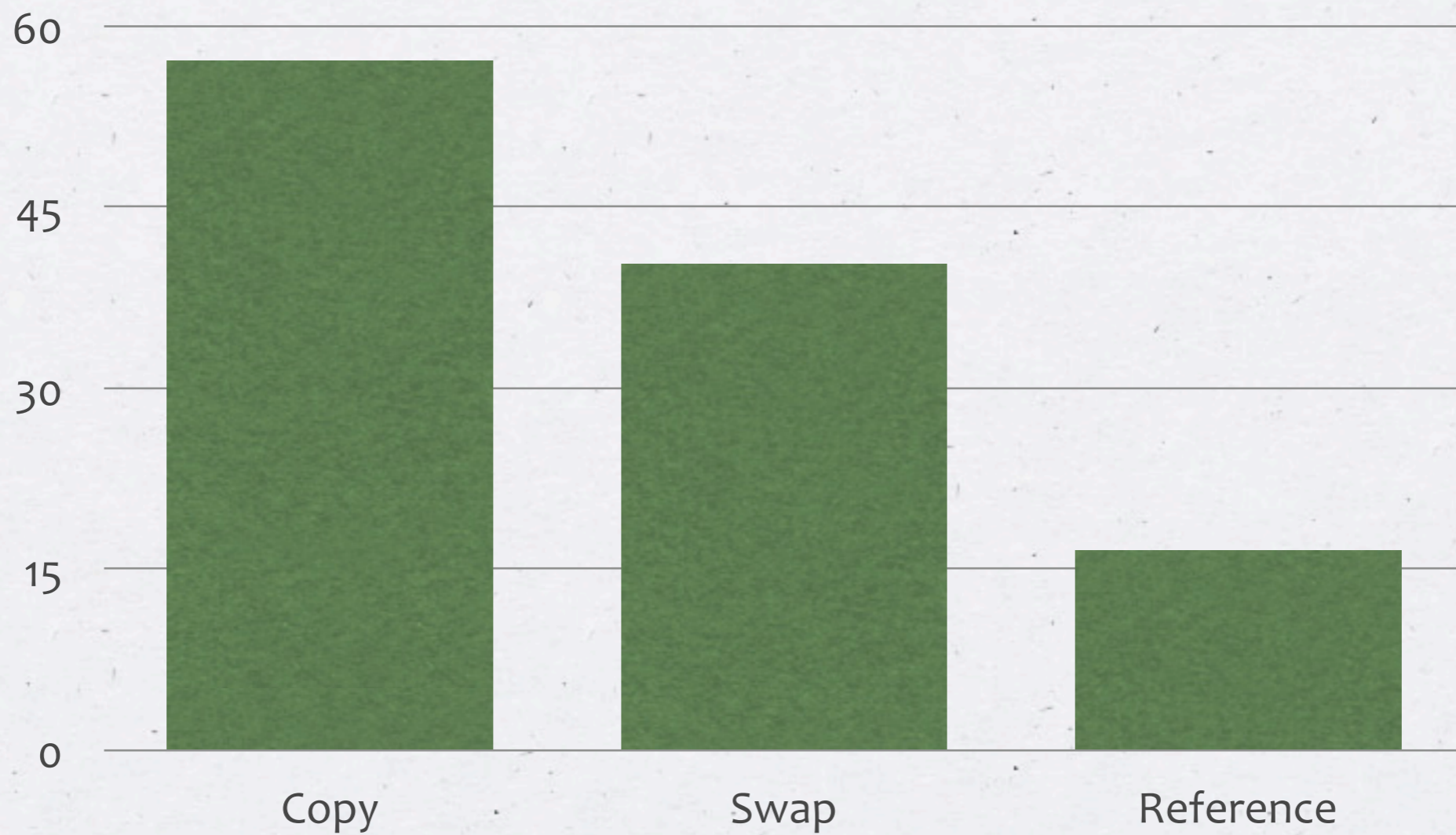
Post-state:

$p = ???$

$q = ???$

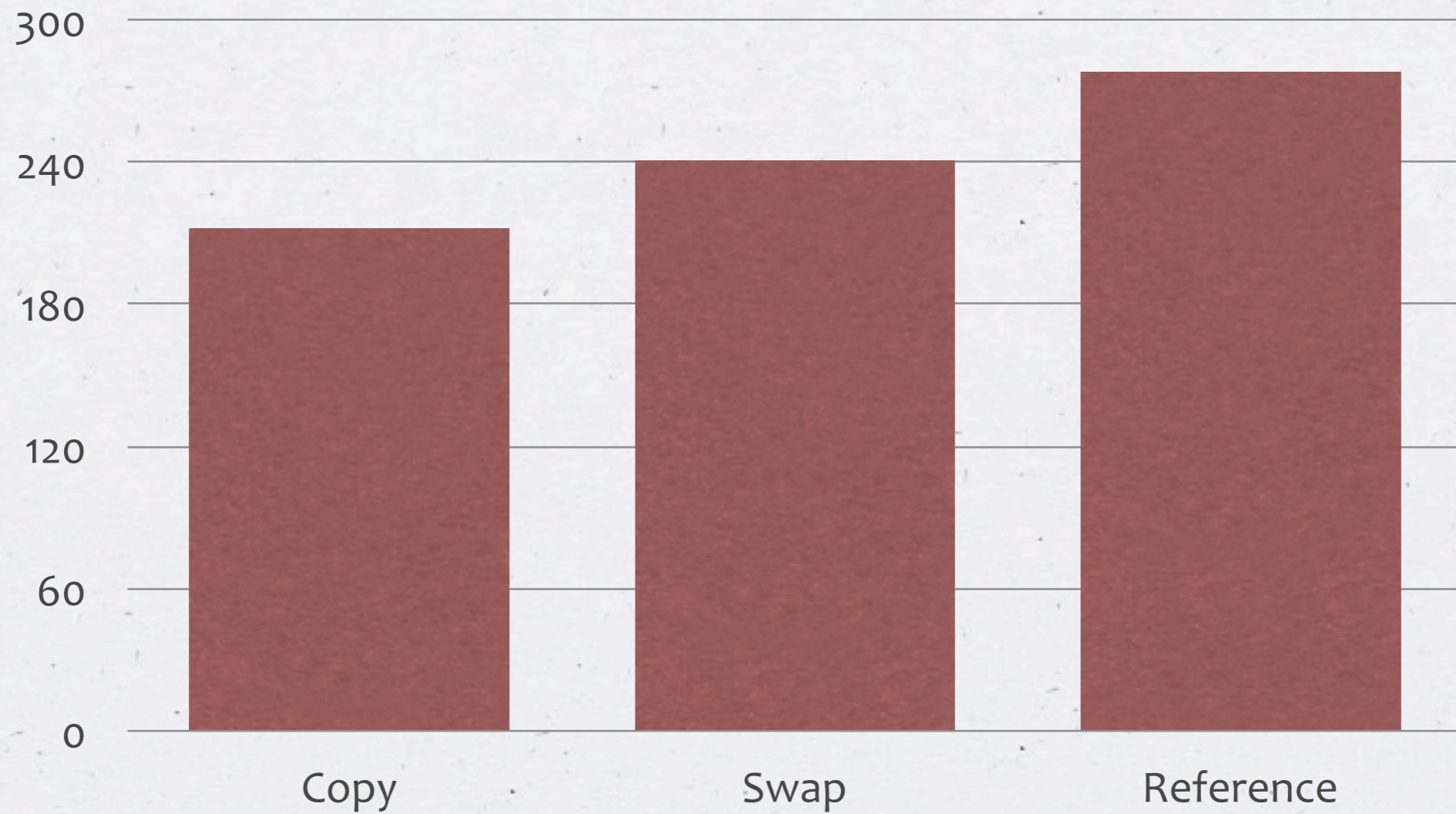
$t = ???$

Percent Correct



Average Time

(to answer correctly)



Future for *swapping*-based OO?

Our case study



West of House

You are standing in an open field west of a white house, with a boarded front door.

There is a small mailbox here.

> examine mailbox

The small mailbox is closed

> open mailbox

Opening the small mailbox reveals a leaflet.

> get leaflet

Taken

>

Parser

Game World

“open the box”



**action = OPEN
subject = PLAYER
object1 = BOX
object2 = NOTHING**

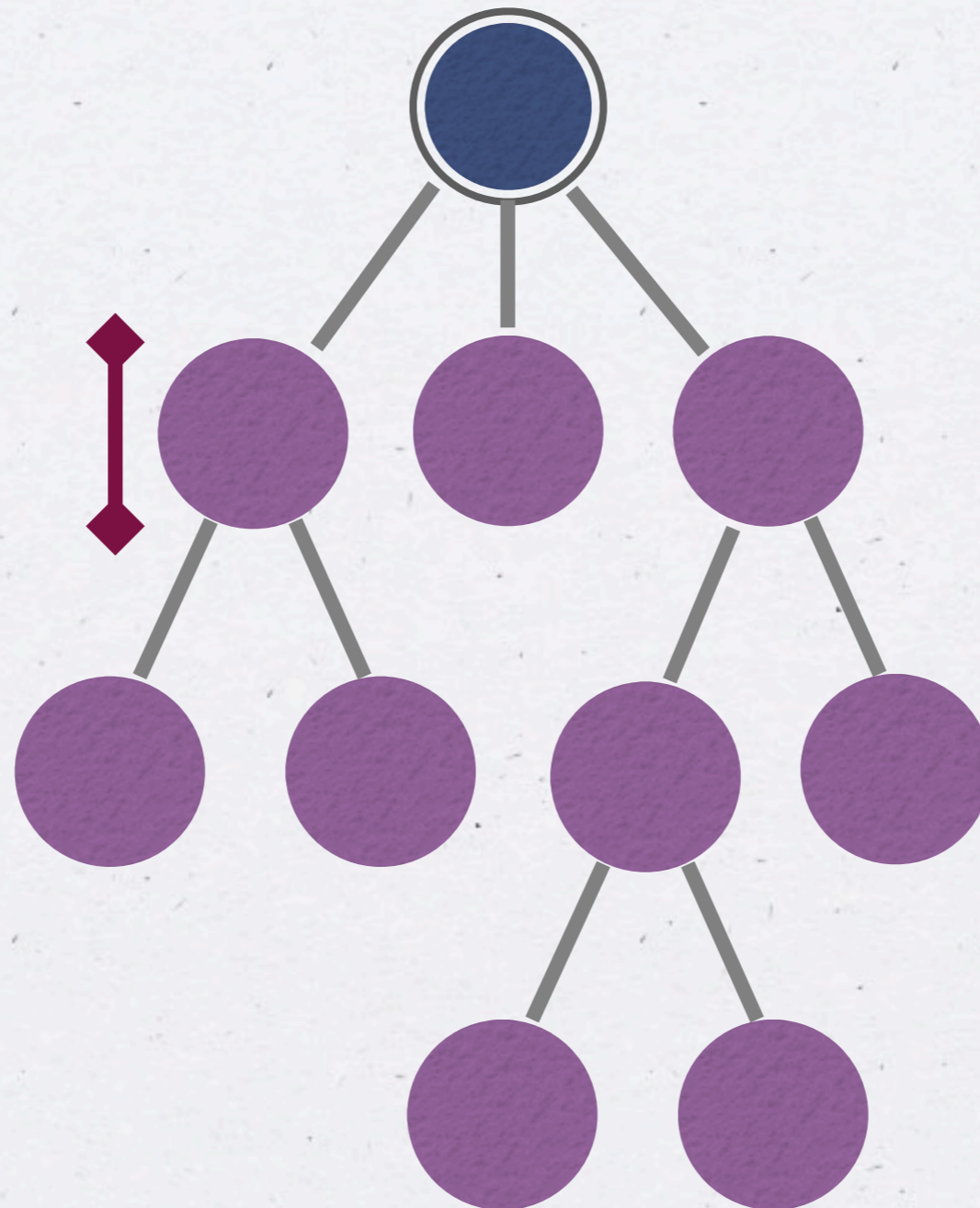
Parser: Tako \approx Java

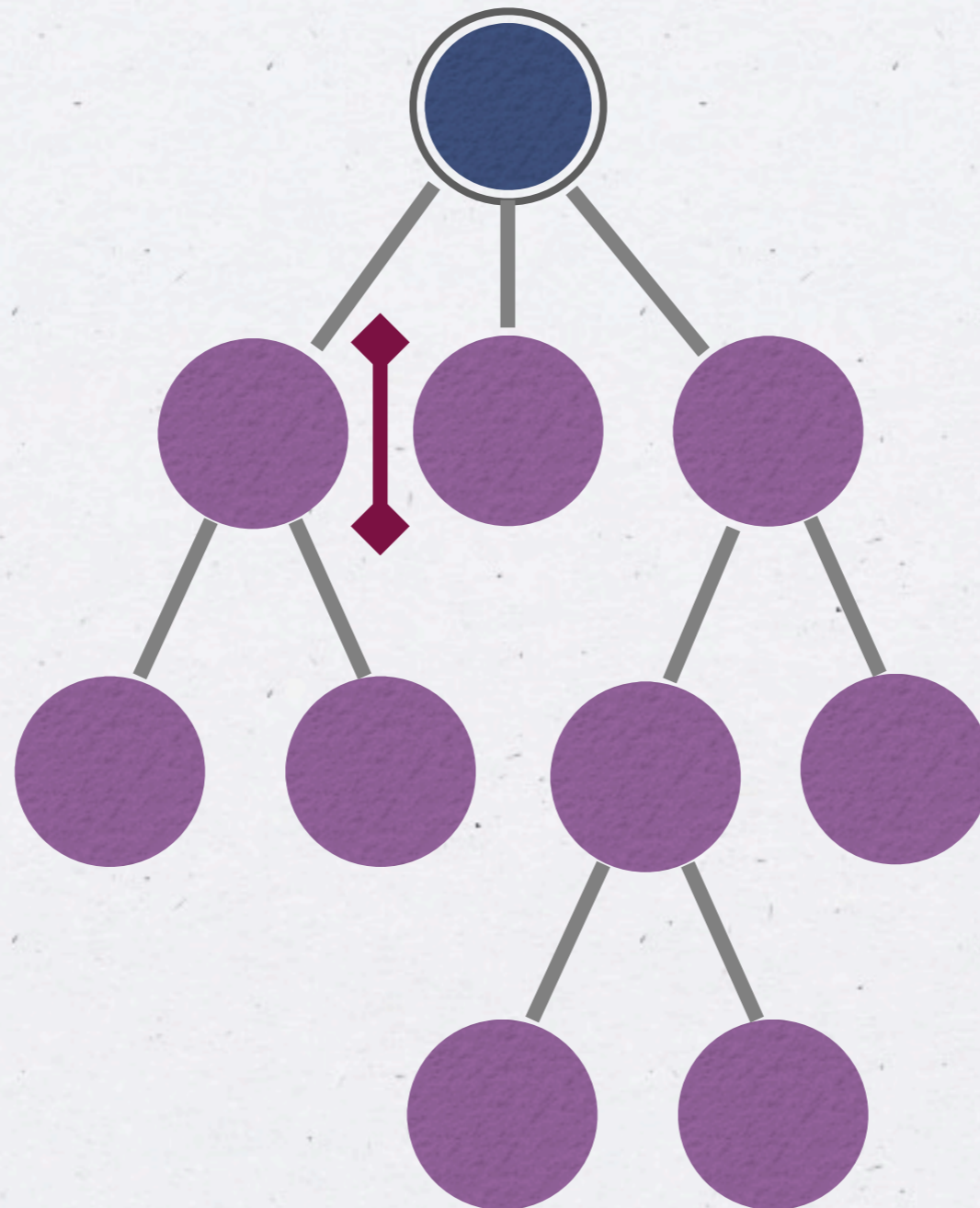
Game World

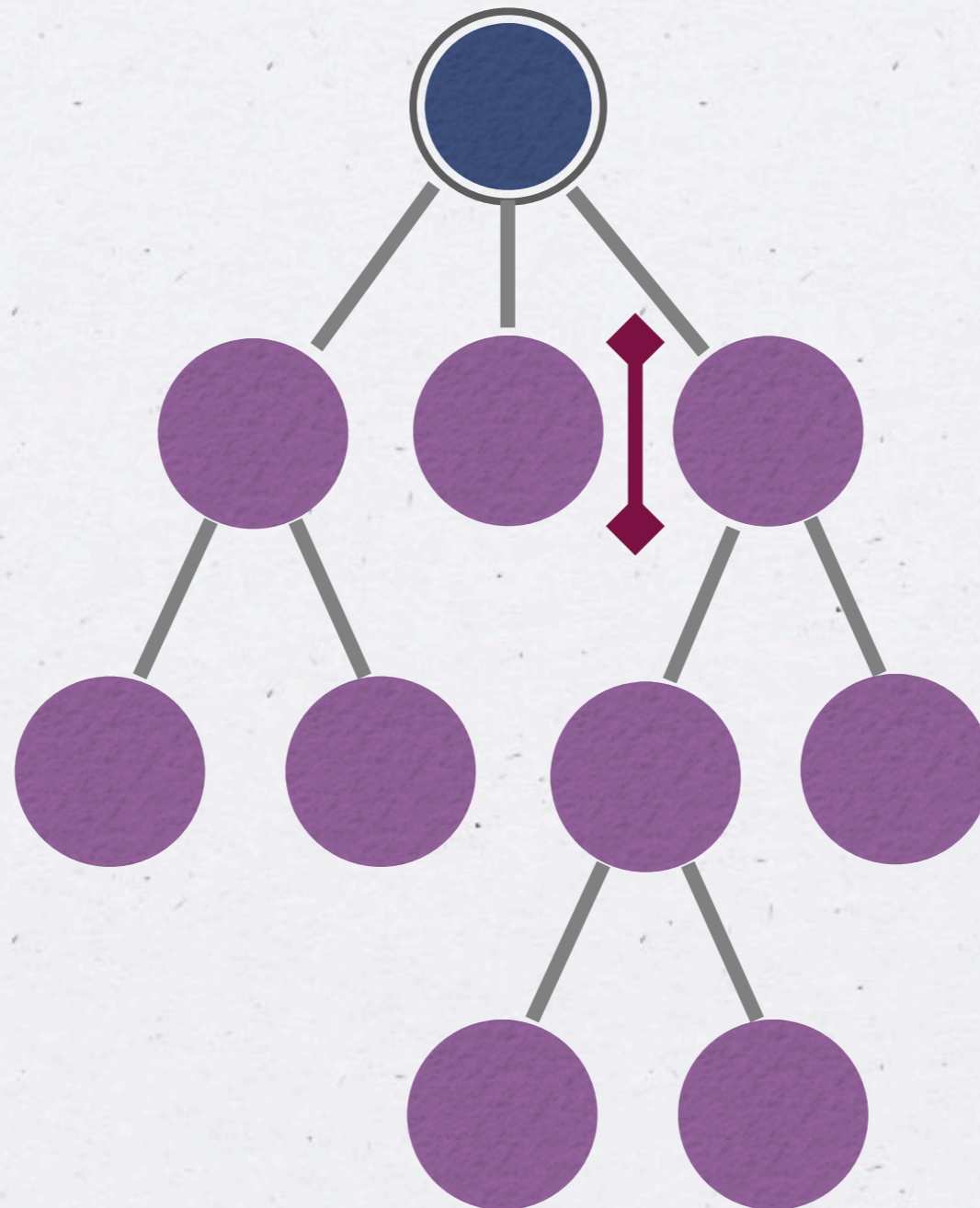


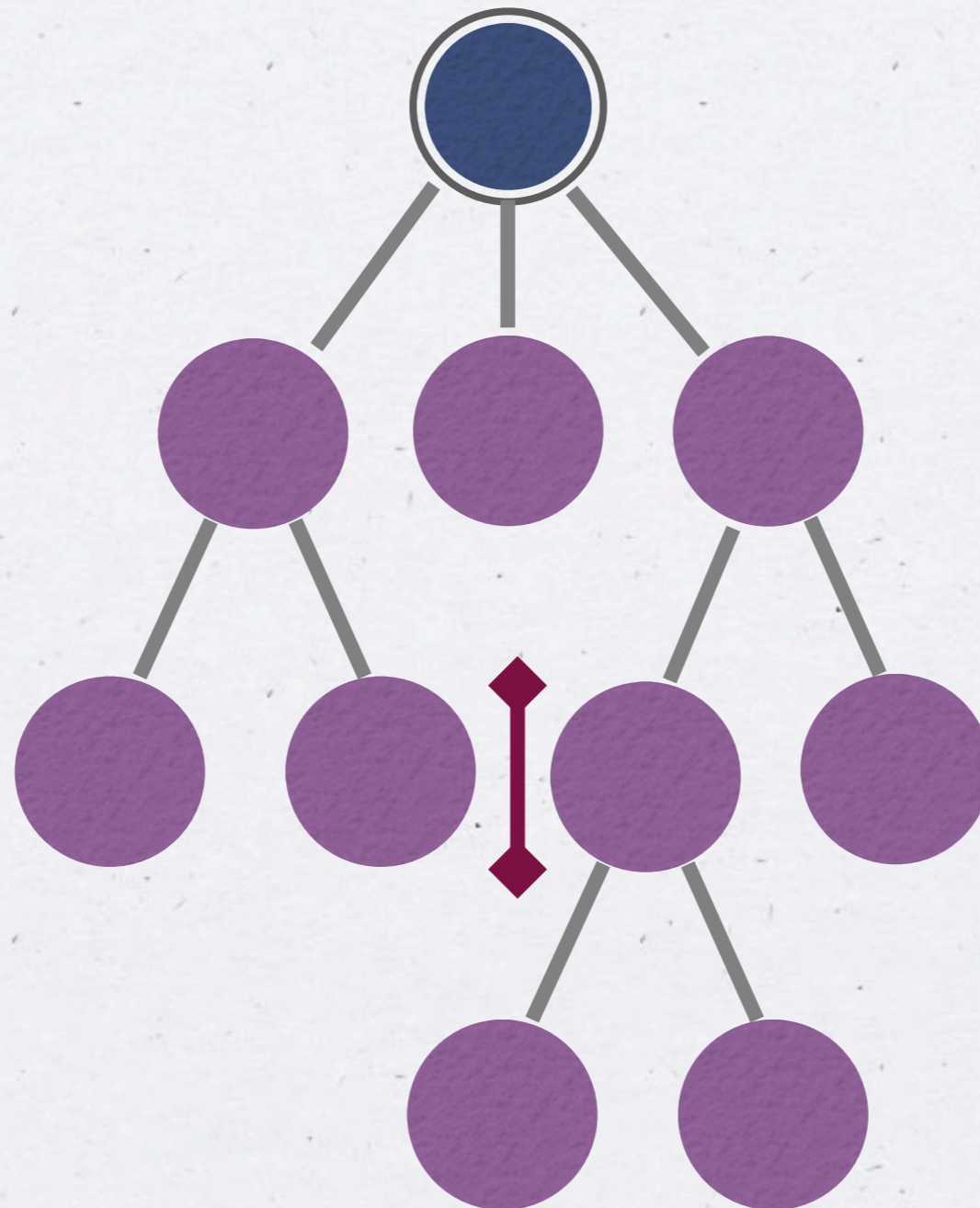
**Tree of
Game
Objects**

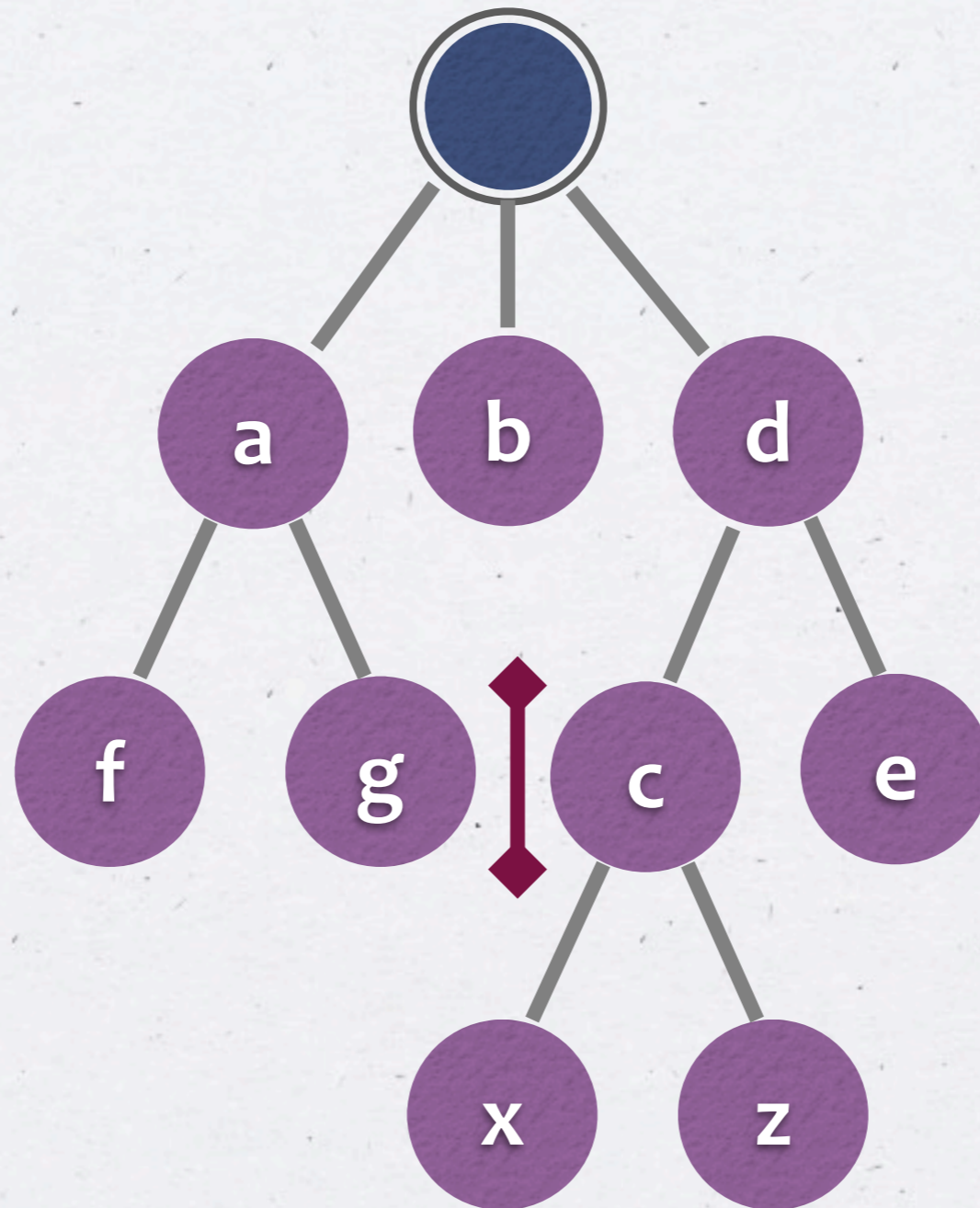
Indexed Tree

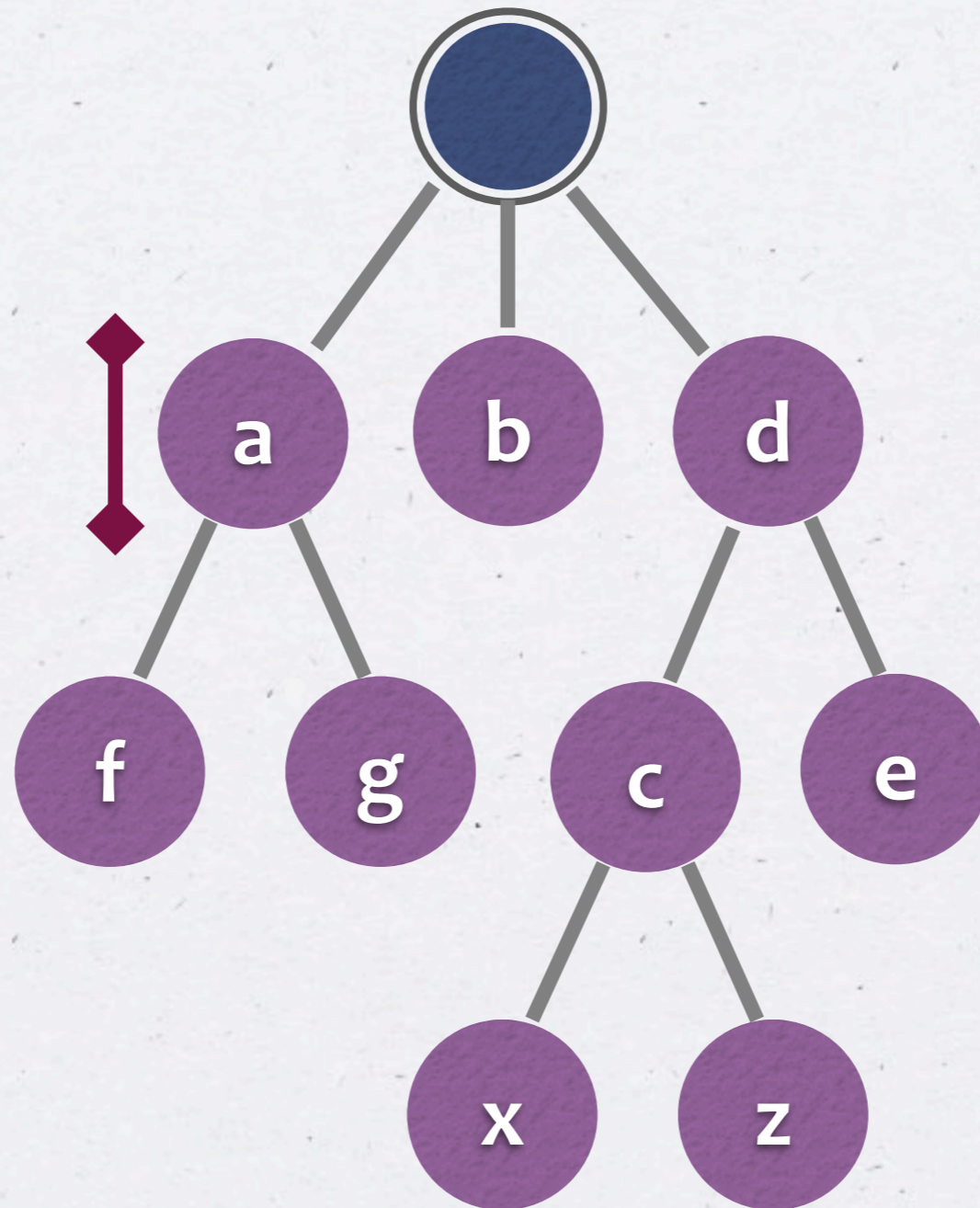












Model

$G = (V, E)$

$\text{rank}(v)$

$\text{ROOT}, \text{CSR} \in V$

$\text{contents}(v)$

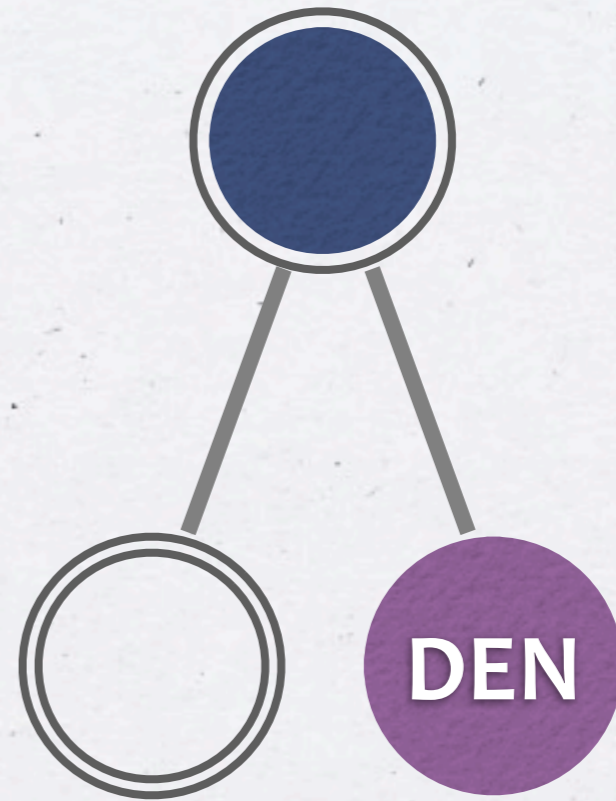
Constraints

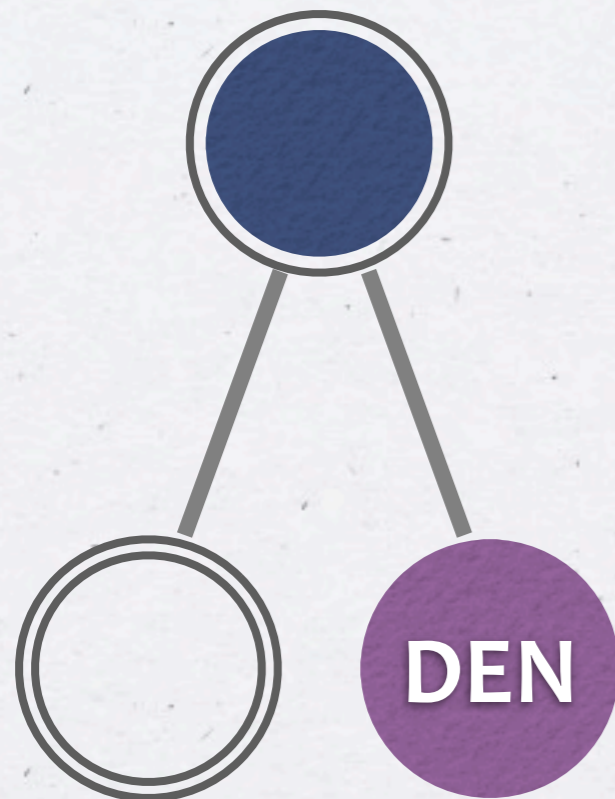
acyclic

consistent rank

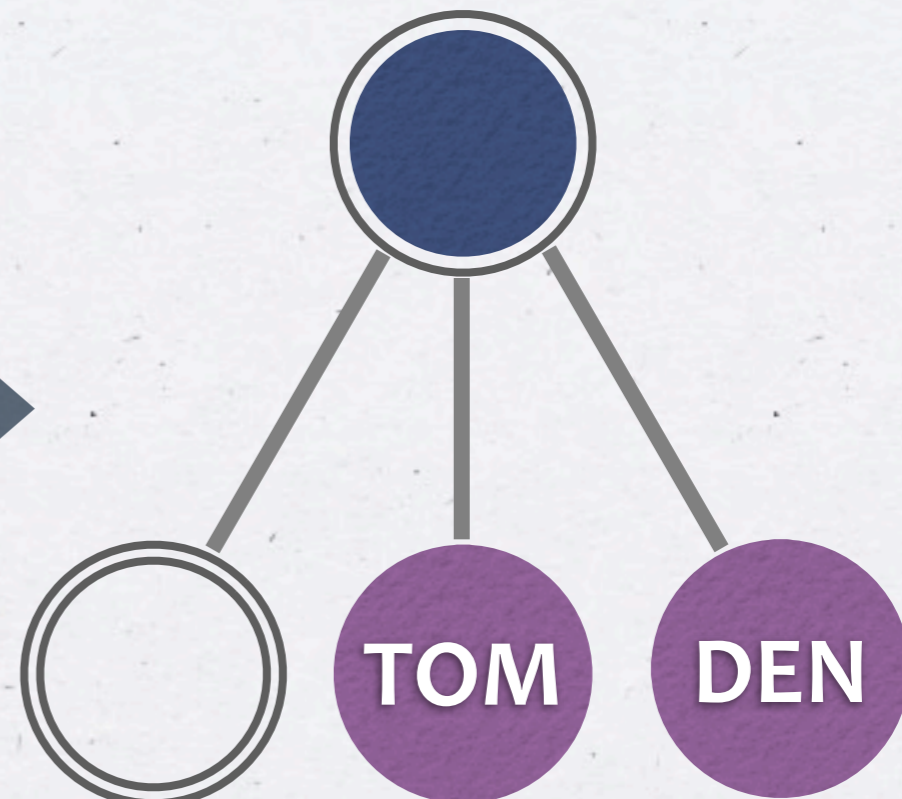


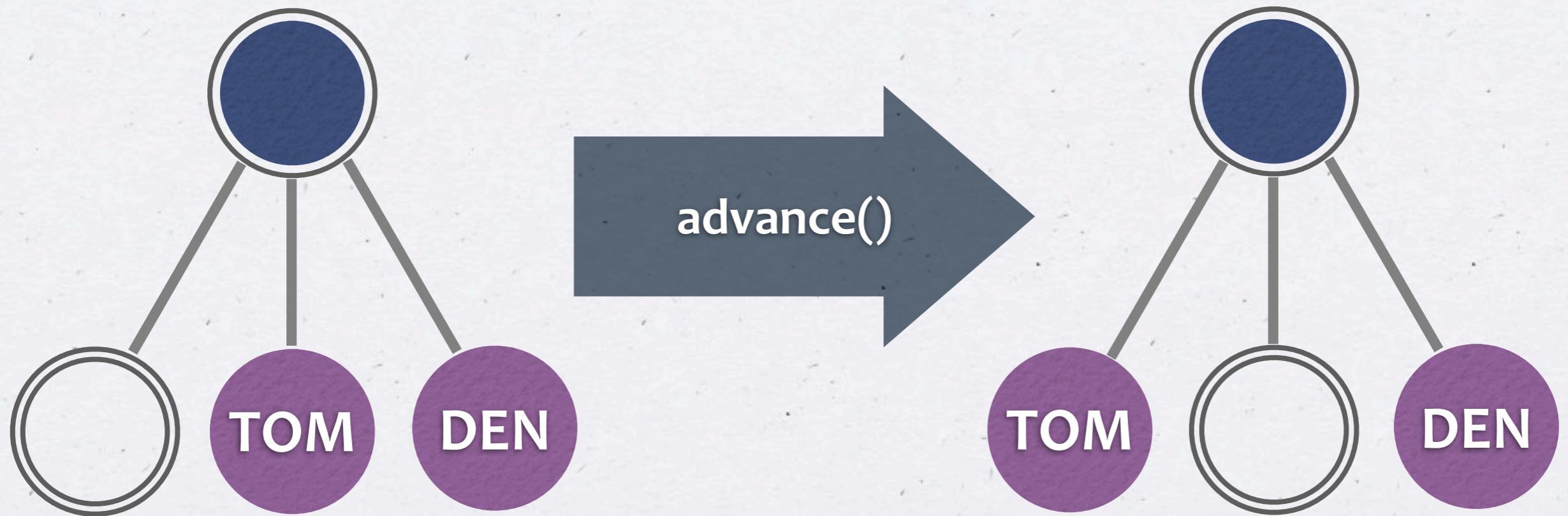
`insert(DEN, den_obj)`

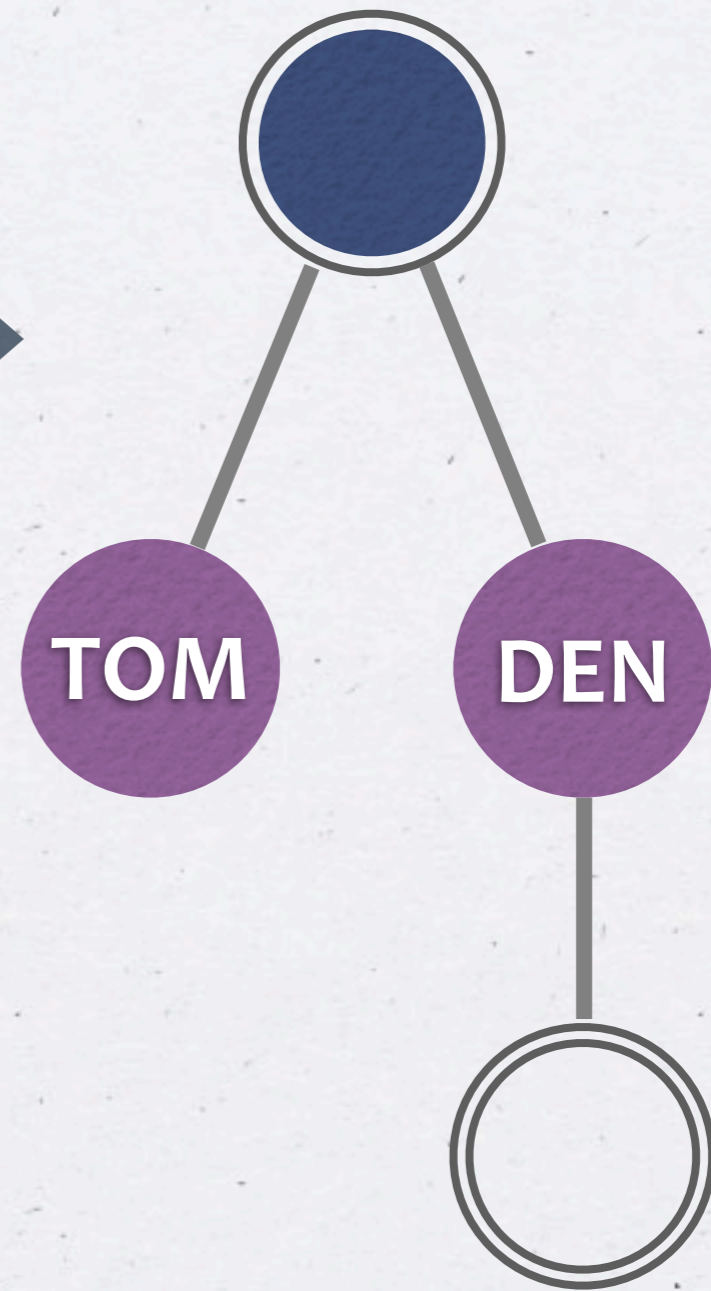
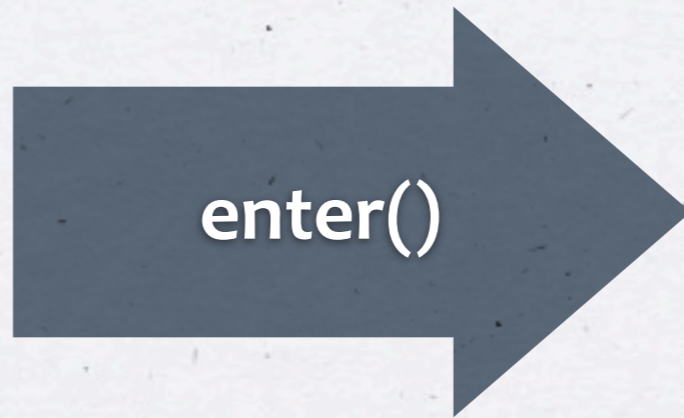
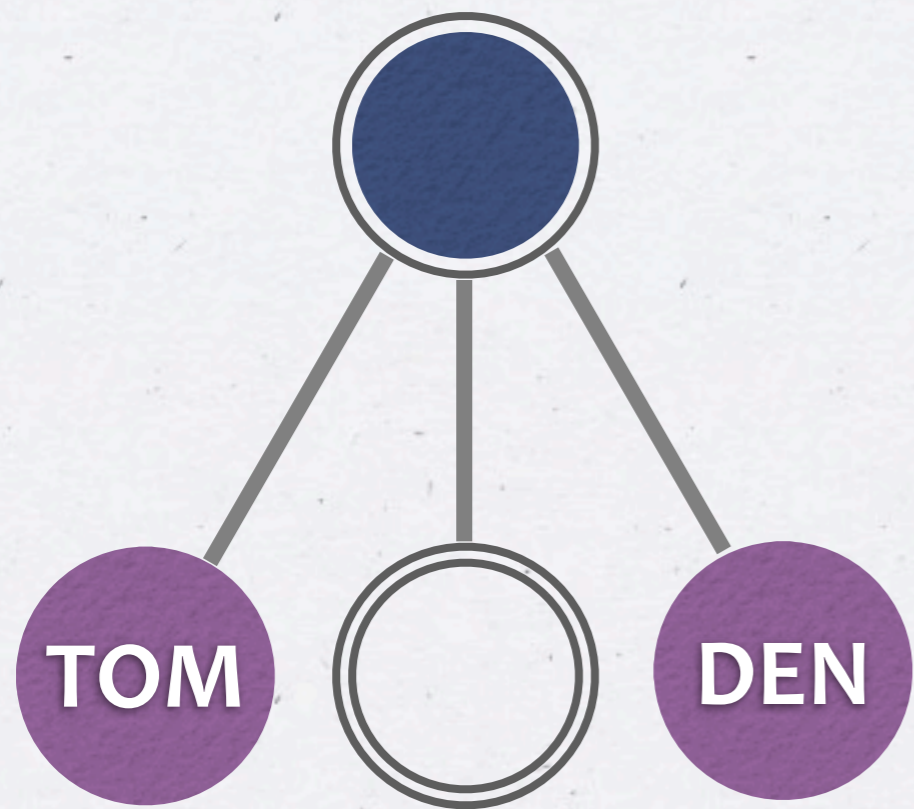


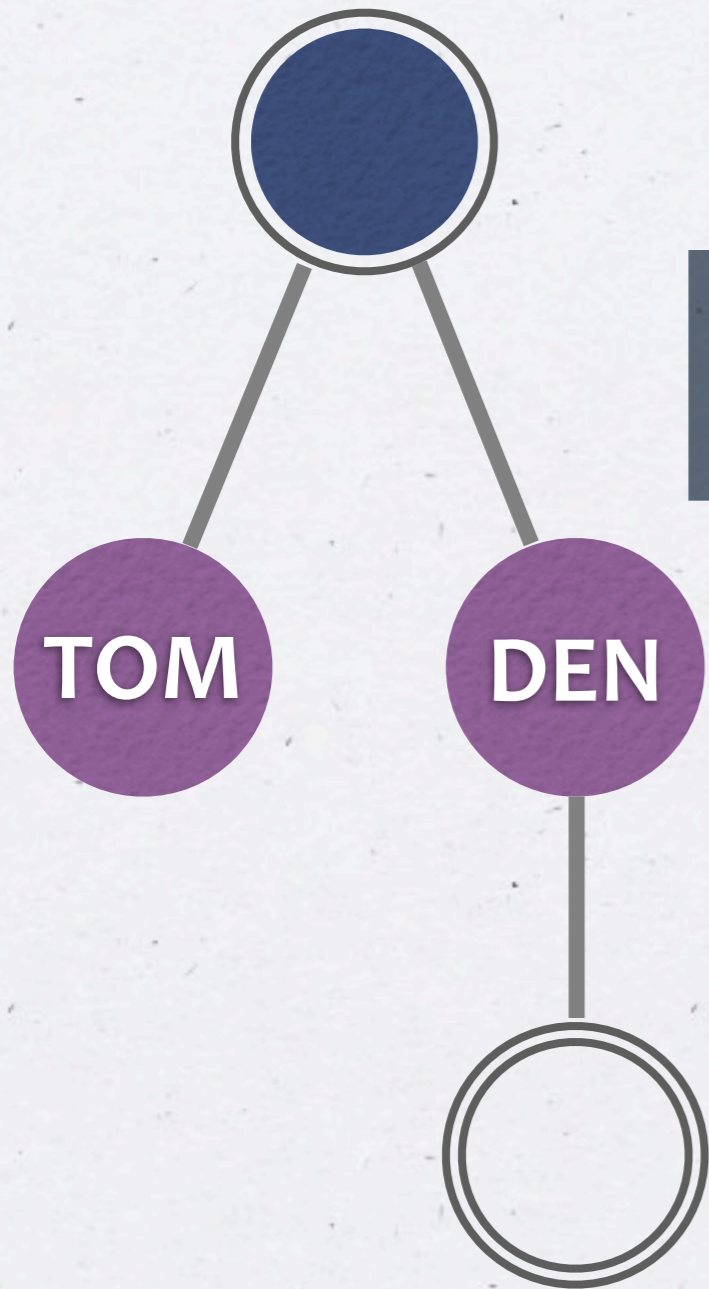


`insert(TOM, tom_obj)`

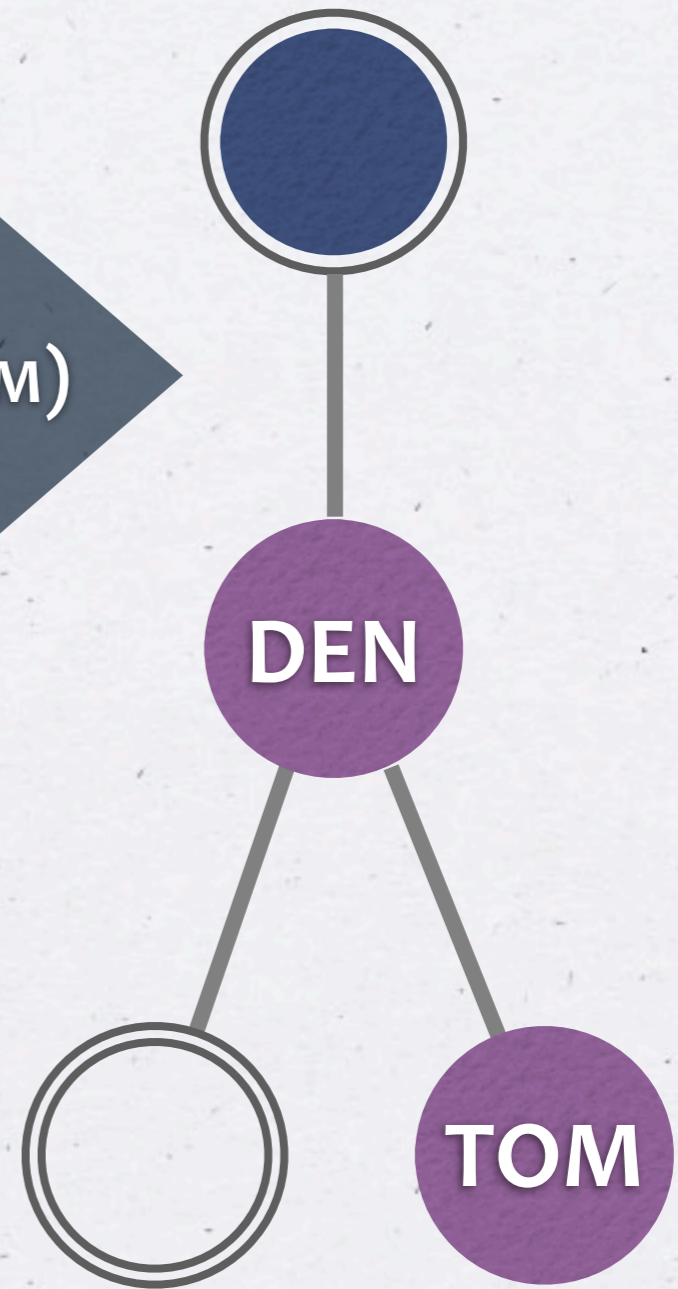








`moveSubtreeToCursor(TOM)`



Game World



**Tree of
Game
Objects**

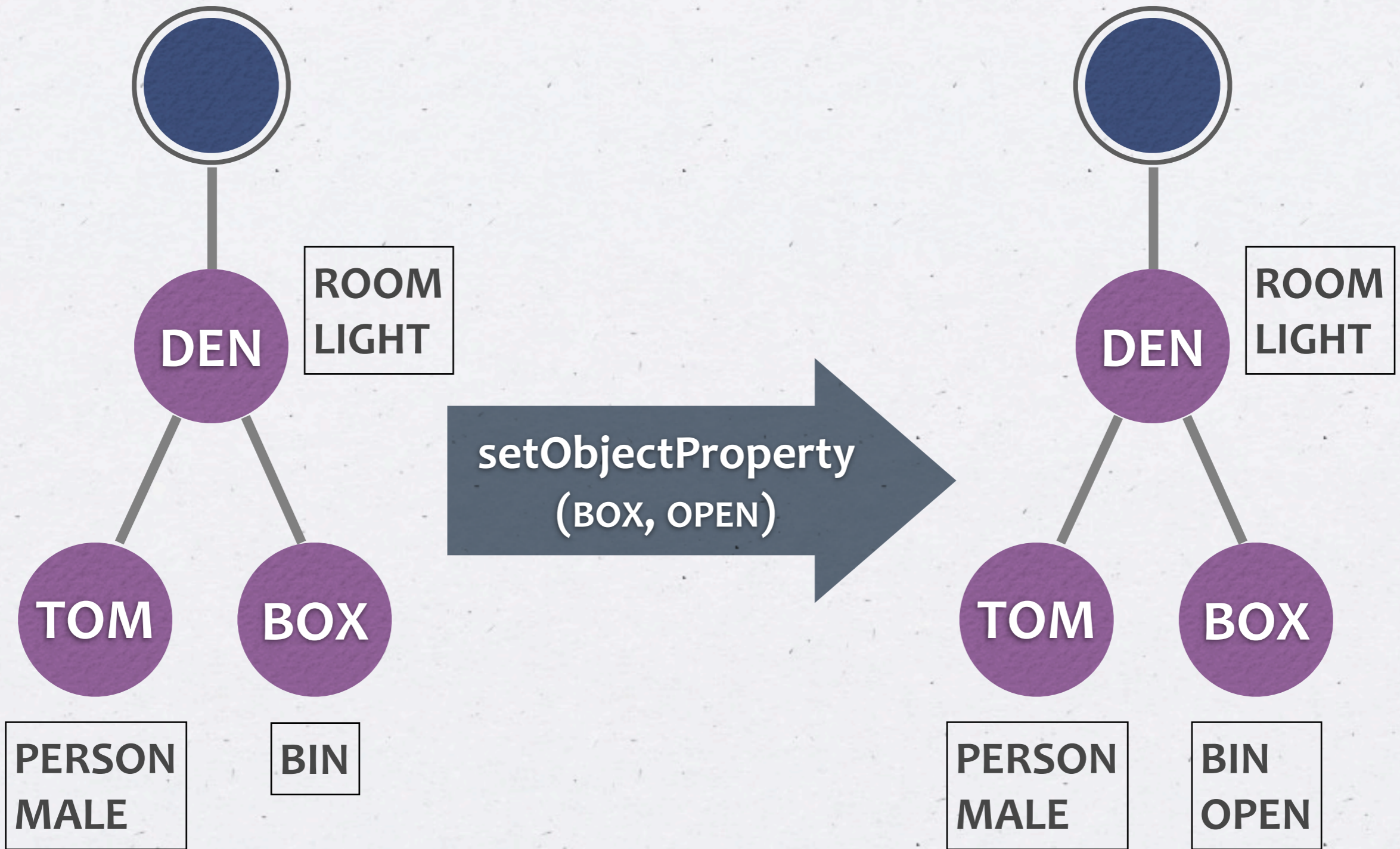
Game World

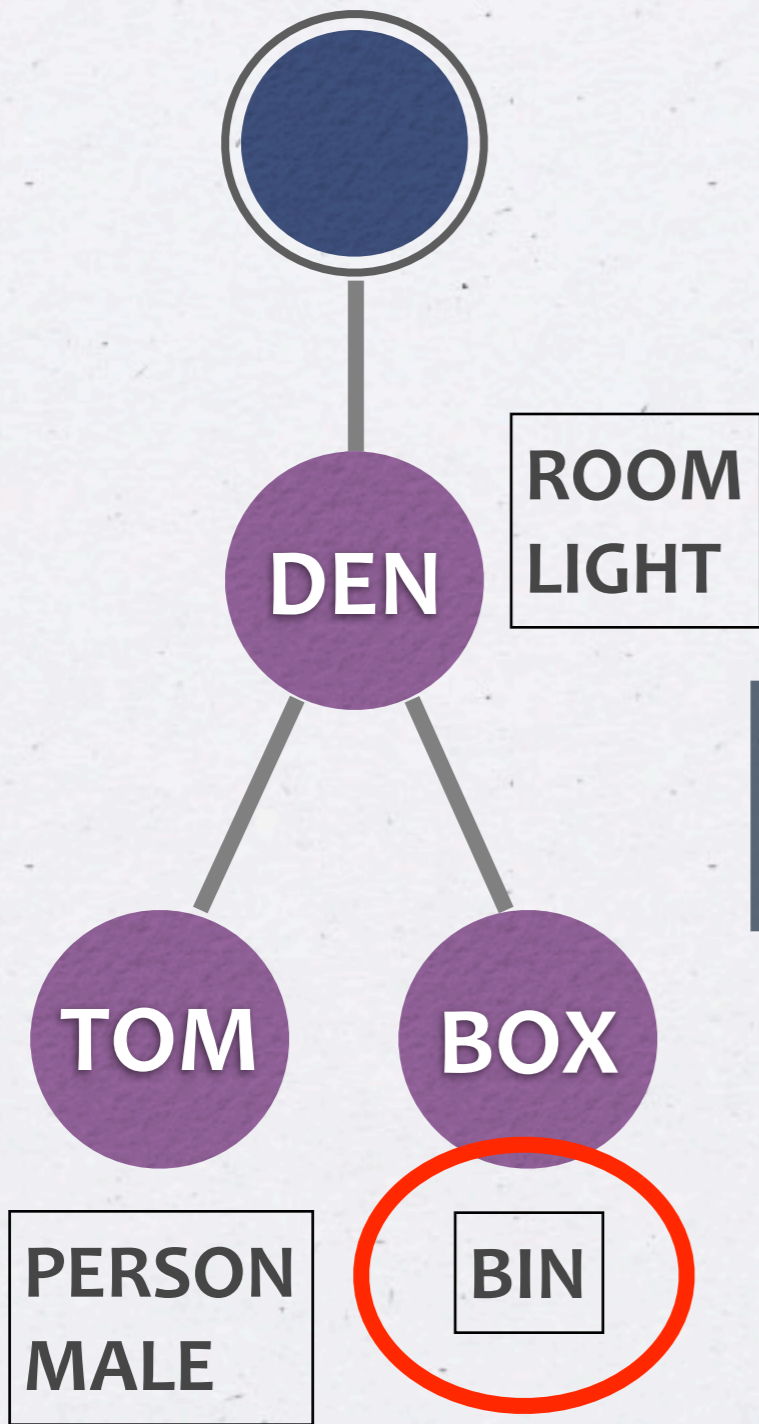
moveObjectIntoSecond(OBJ1, OBJ2)

moveObjectBeforeSecond(OBJ1, OBJ2)

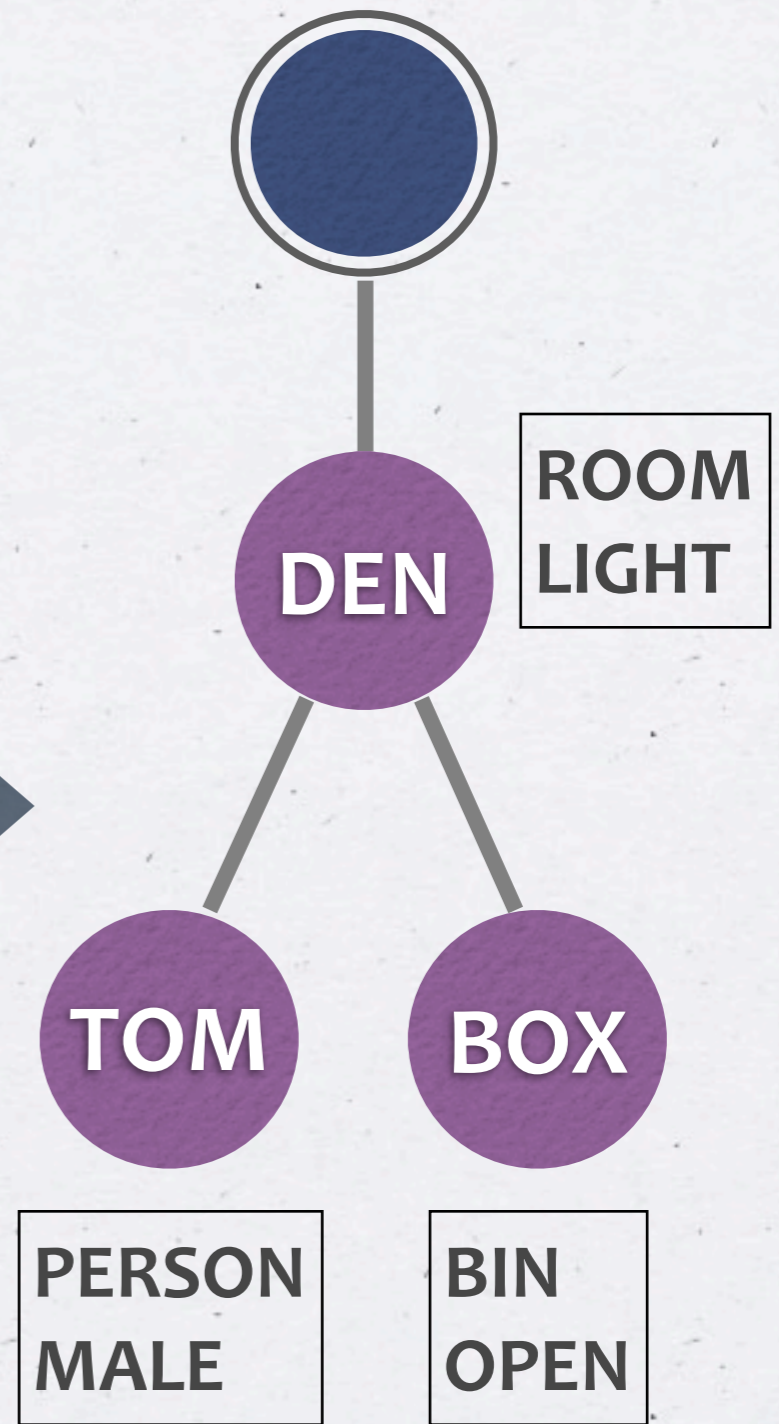
updateObjectProperty(OBJ, PROP)

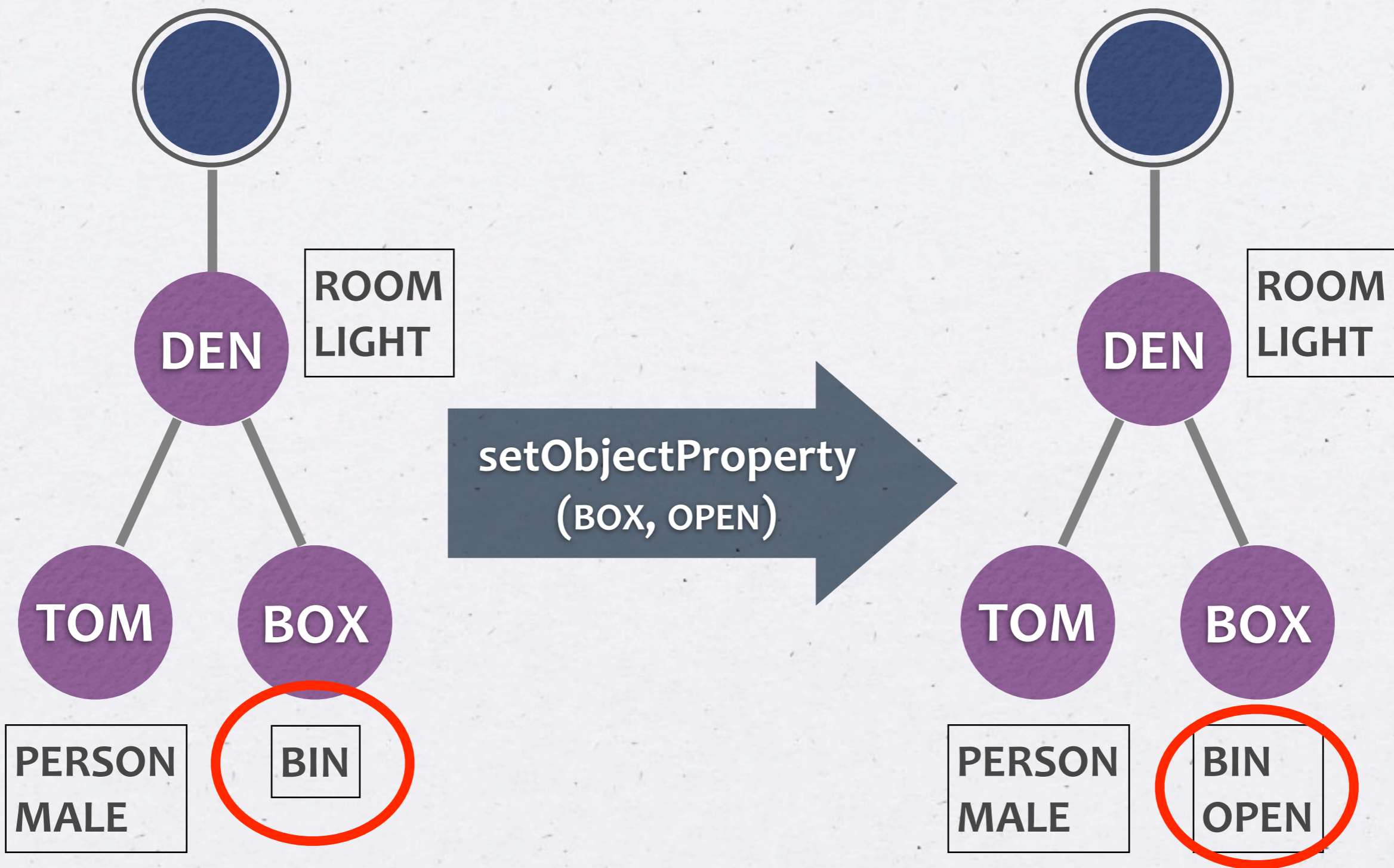
objectHasProperty(OBJ, PROP)

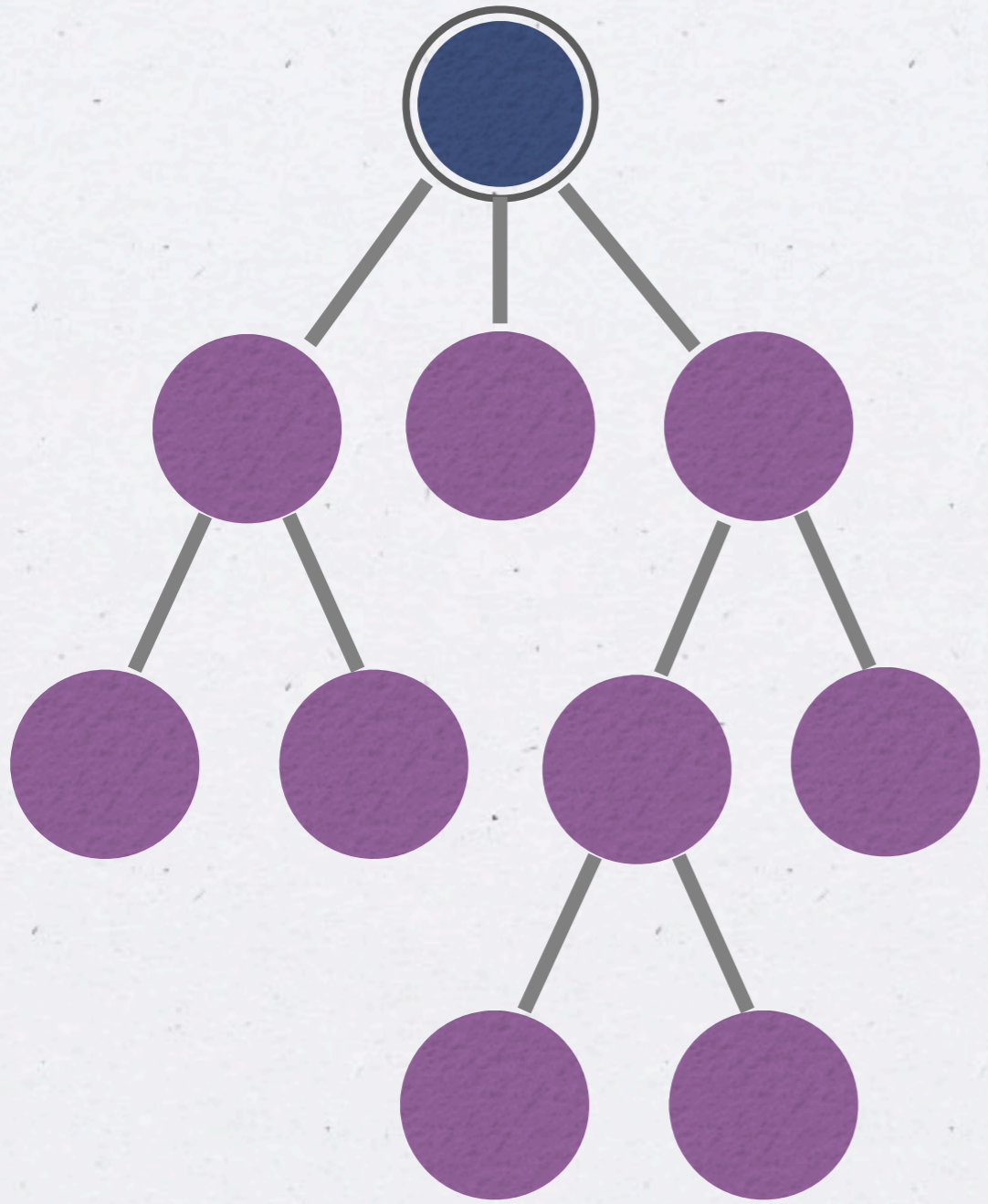


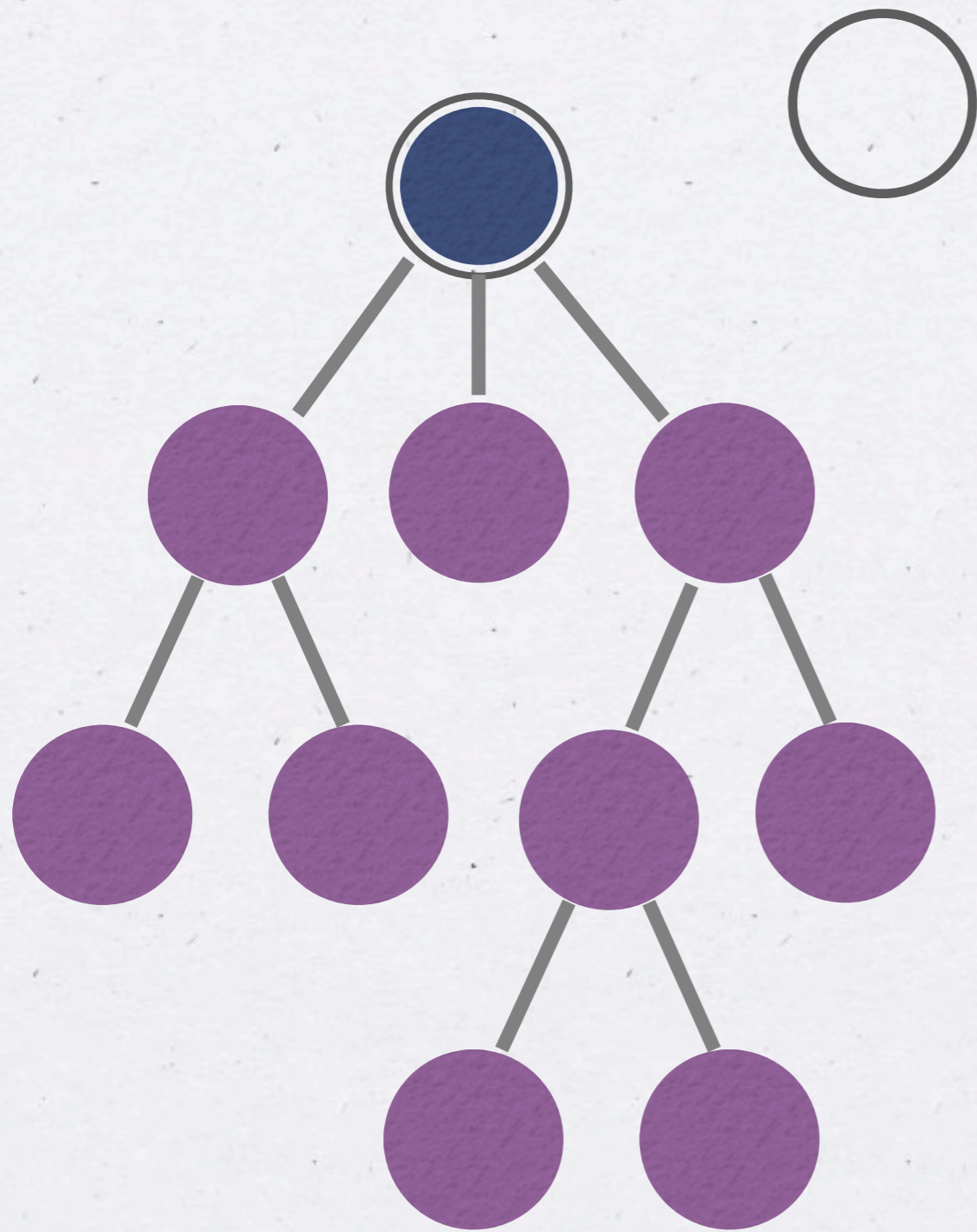


setObjectProperty
(BOX, OPEN)

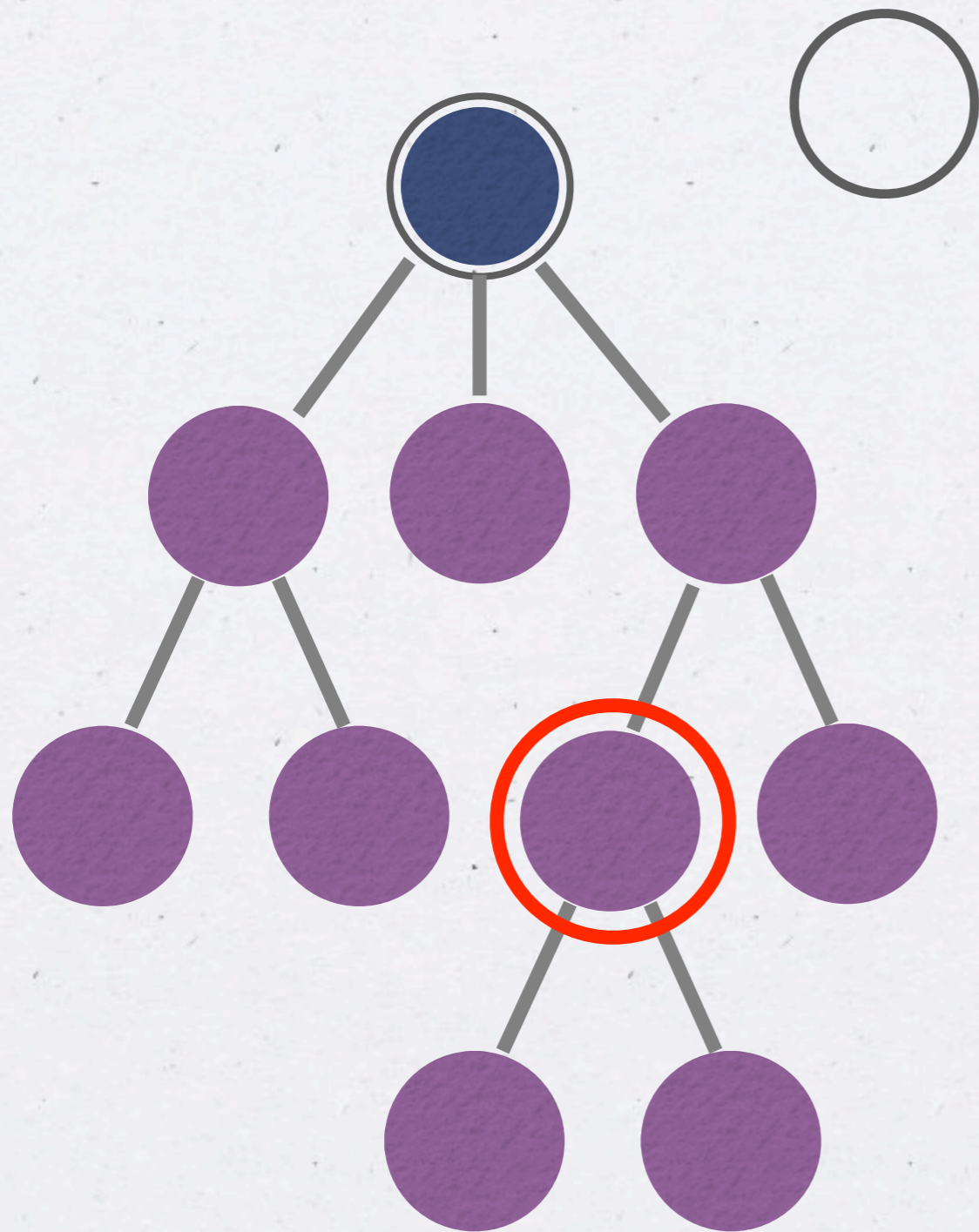




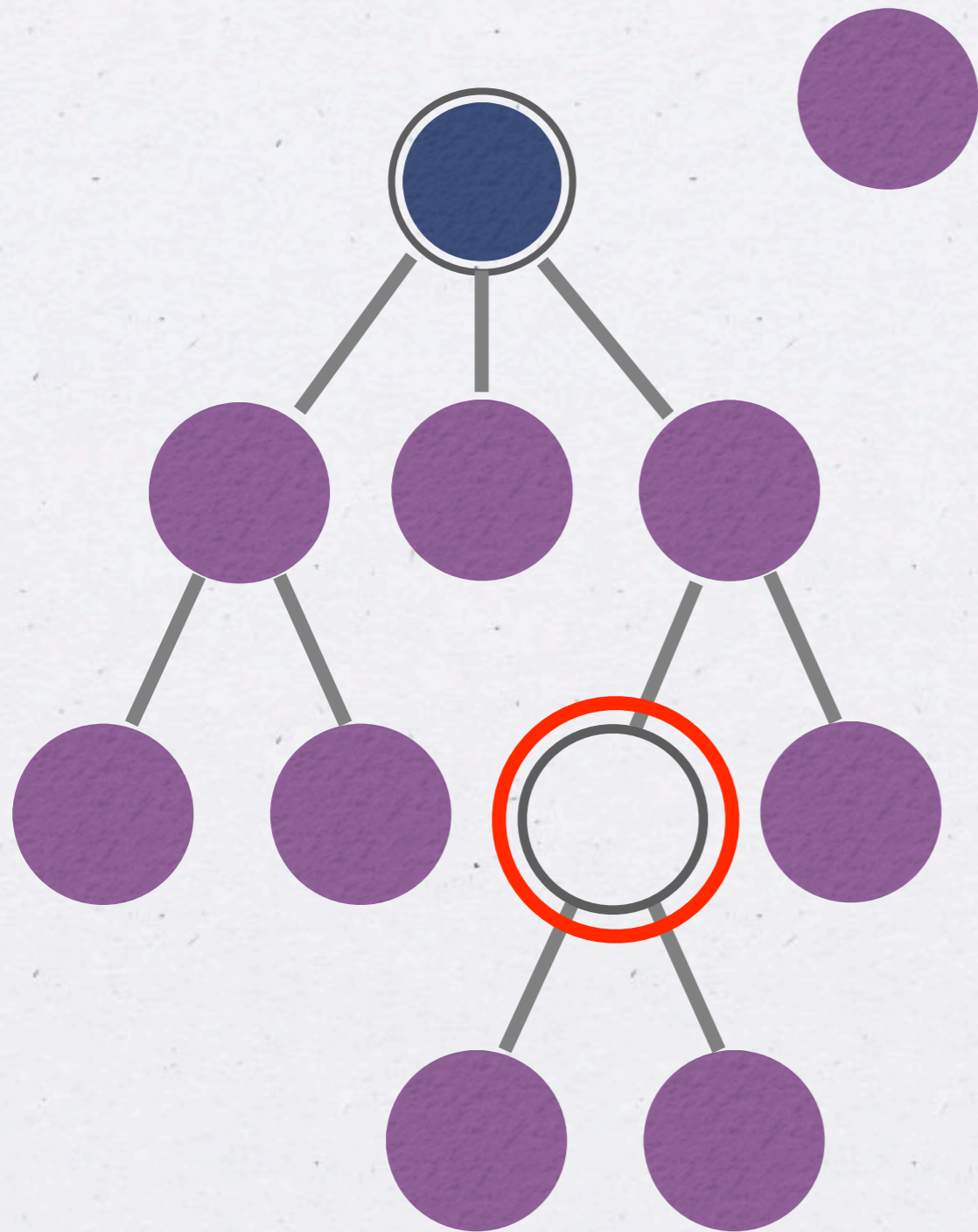




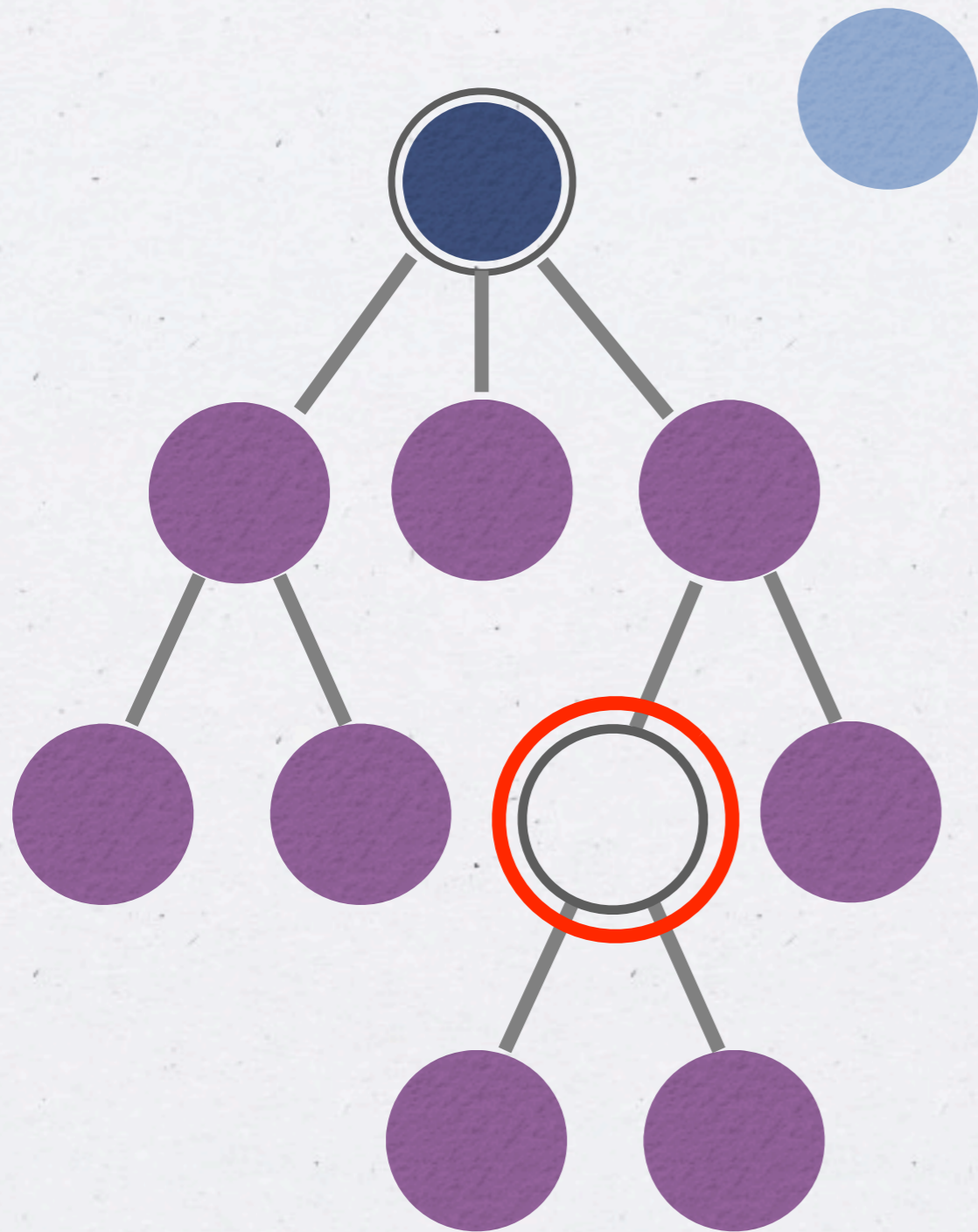
1. Create dummy node



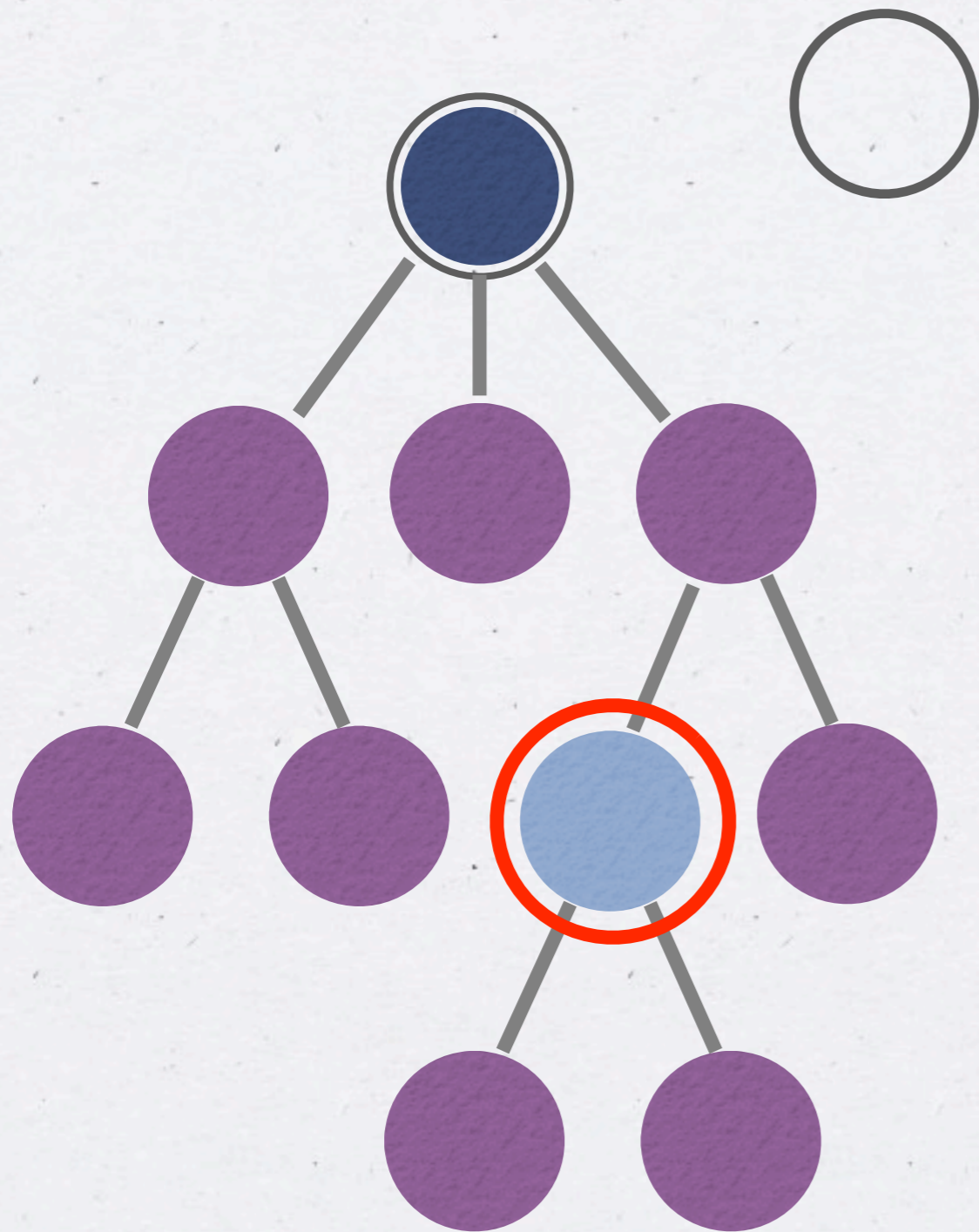
- 1. Create dummy node**
- 2. Go to target**



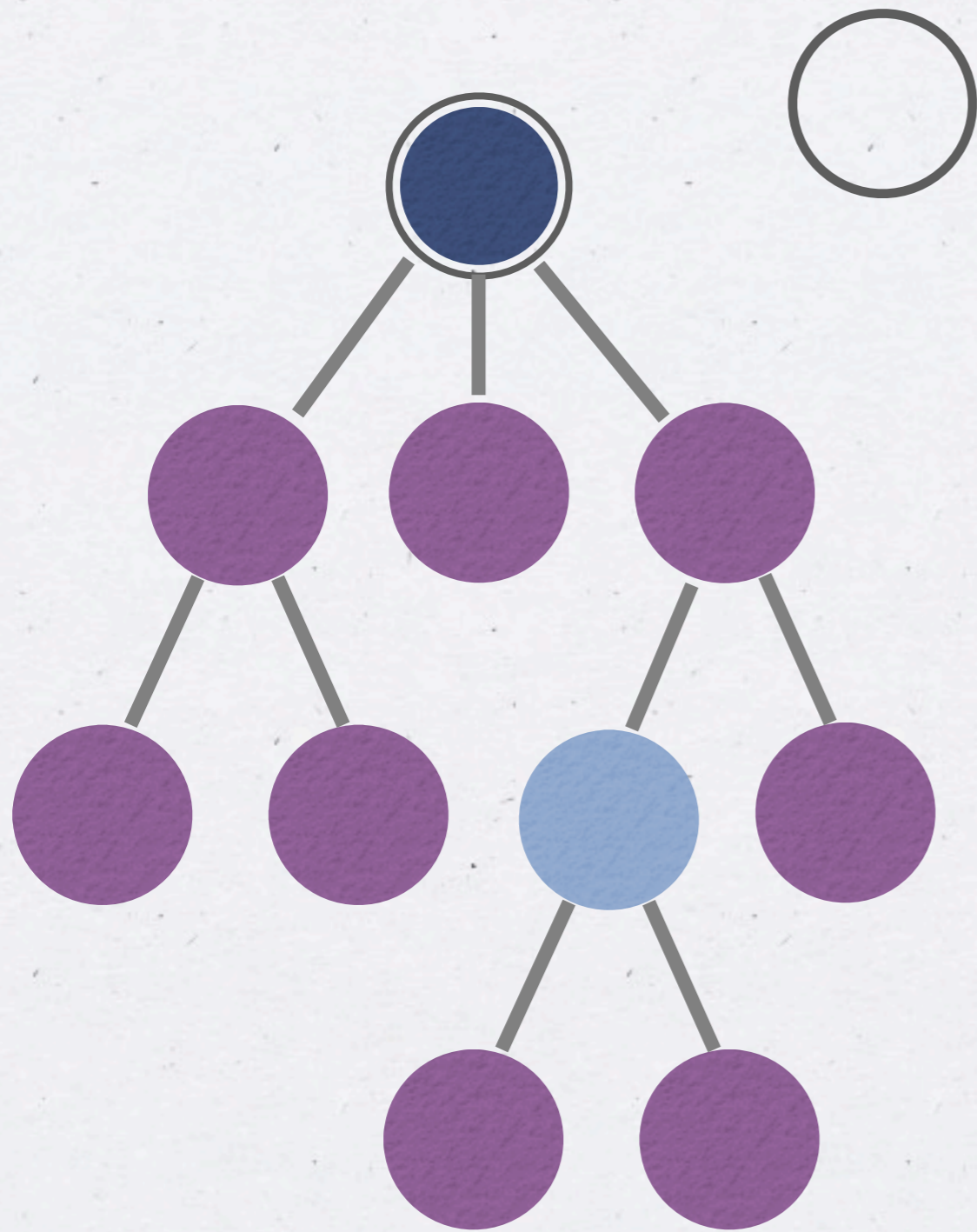
- 1. Create dummy node**
- 2. Go to target**
- 3. Swap node out**



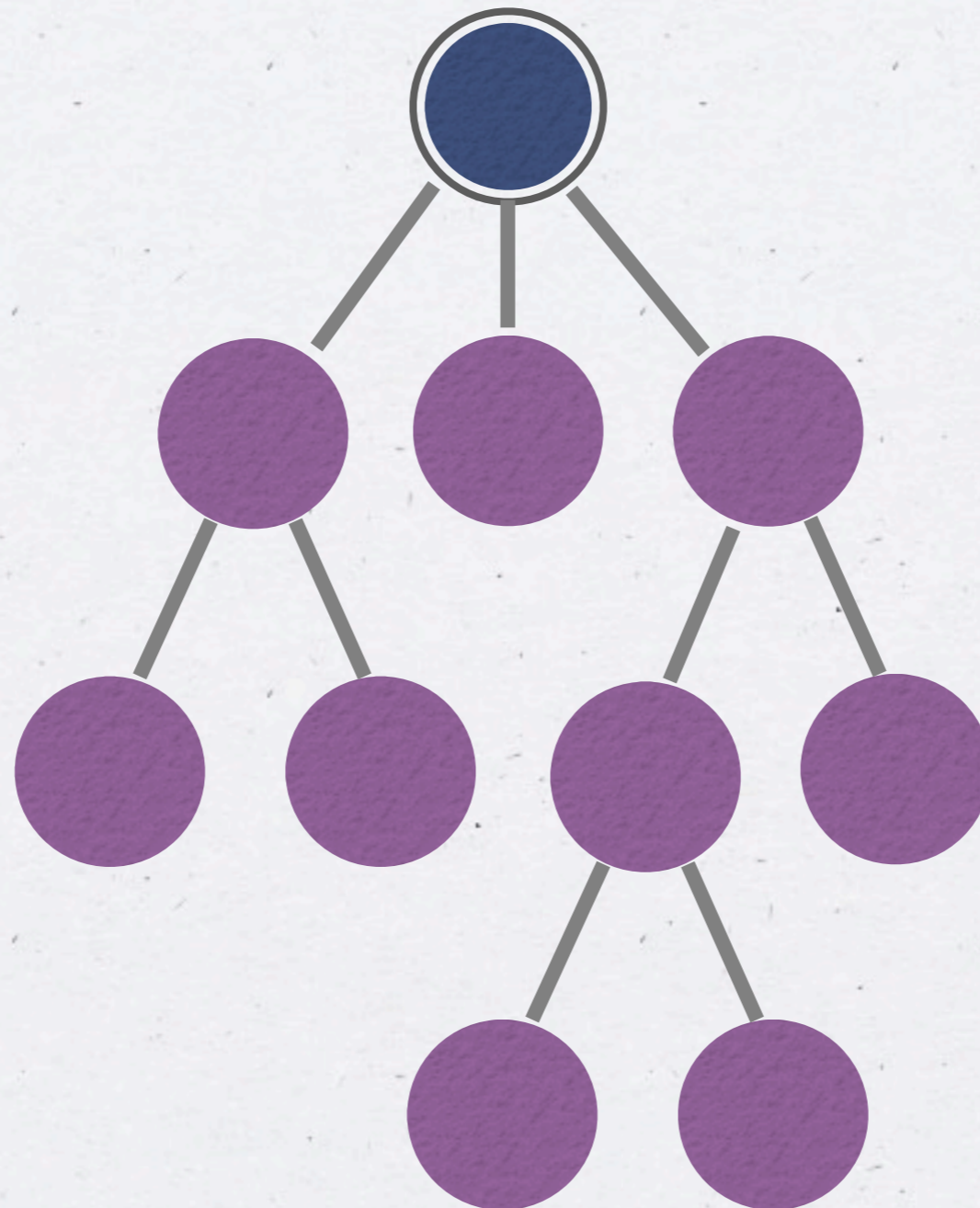
- 1. Create dummy node**
- 2. Go to target**
- 3. Swap node out**
- 4. Modify node**

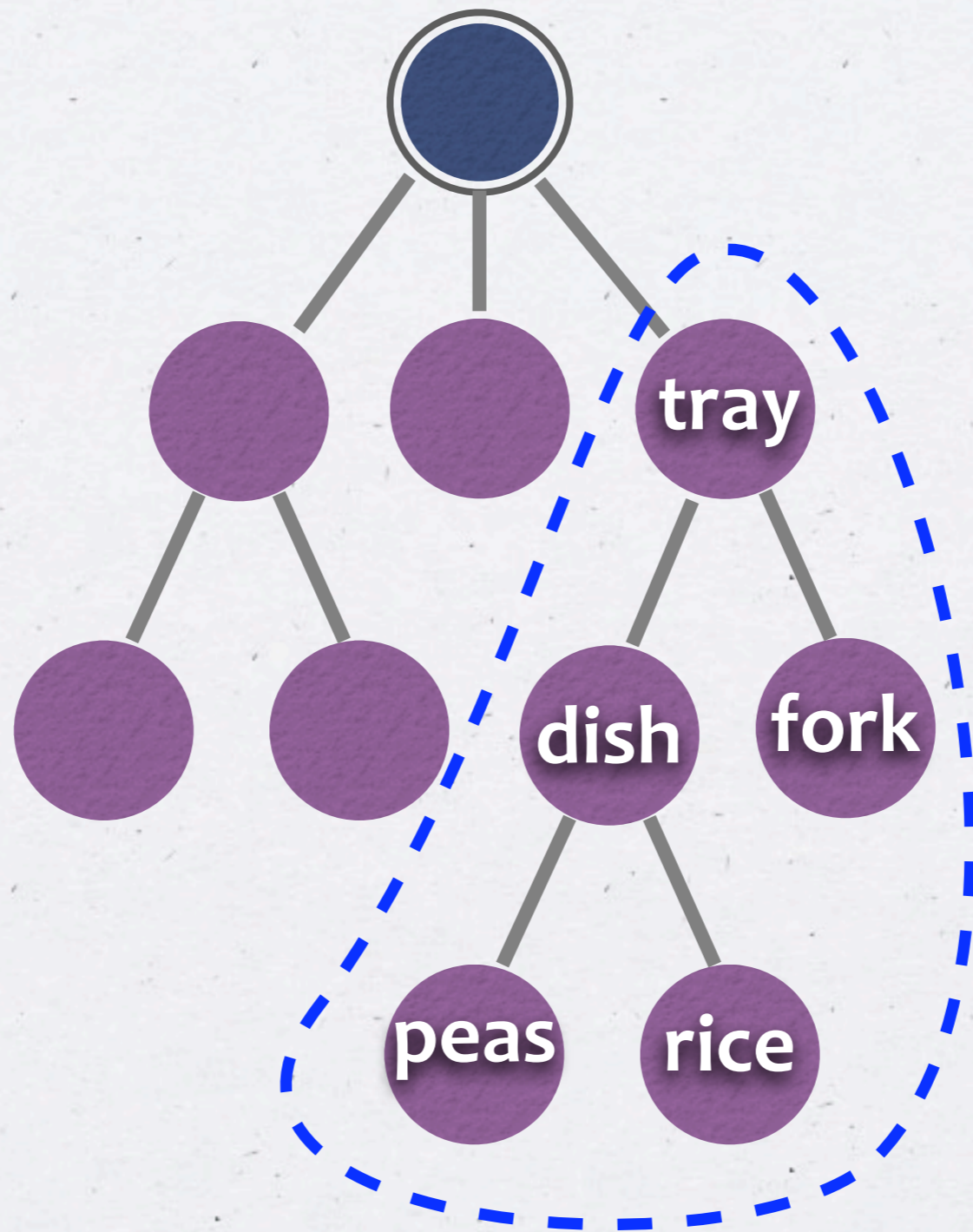


- 1. Create dummy node**
- 2. Go to target**
- 3. Swap node out**
- 4. Modify node**
- 5. Swap node in**



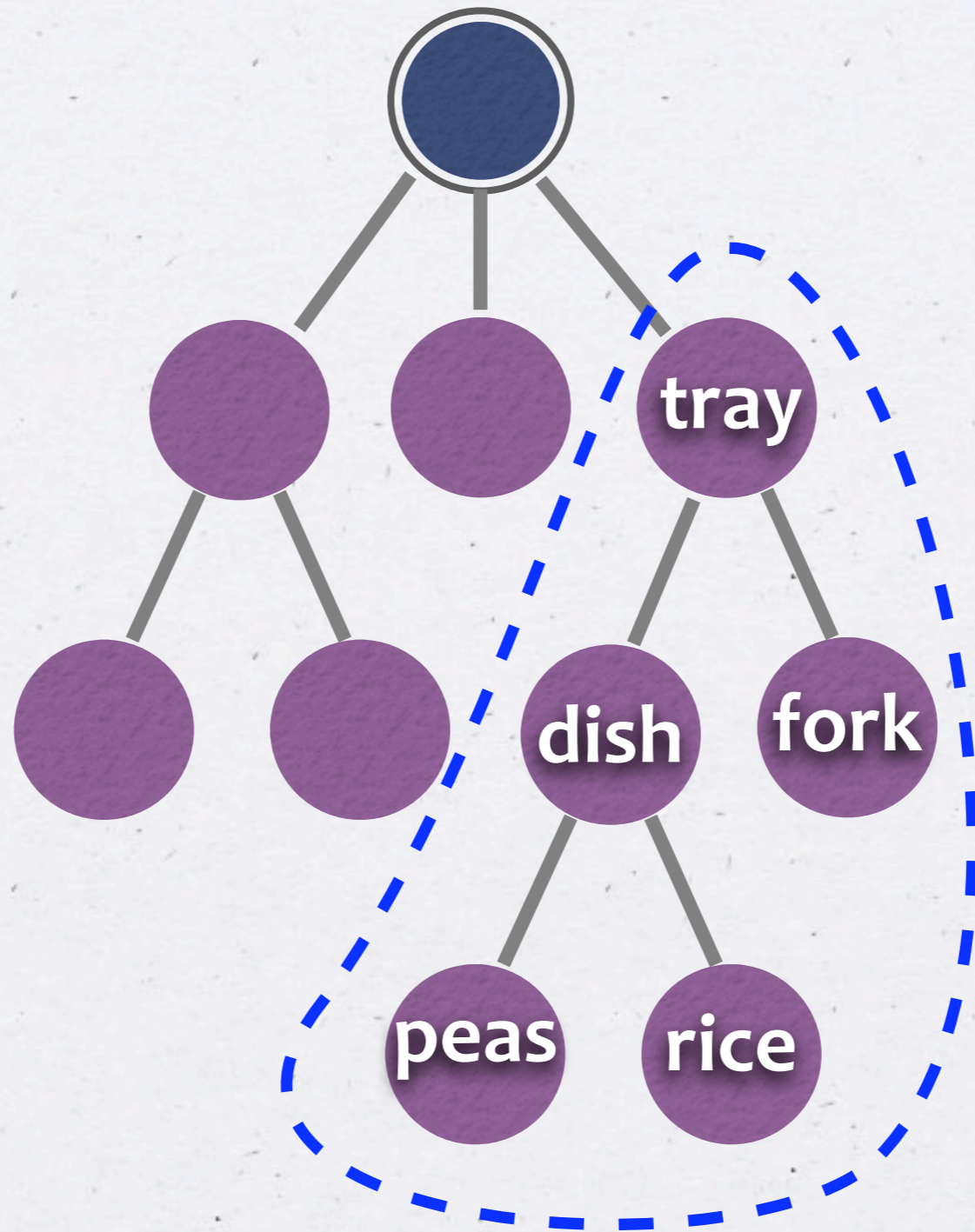
- 1. Create dummy node**
- 2. Go to target**
- 3. Swap node out**
- 4. Modify node**
- 5. Swap node in**





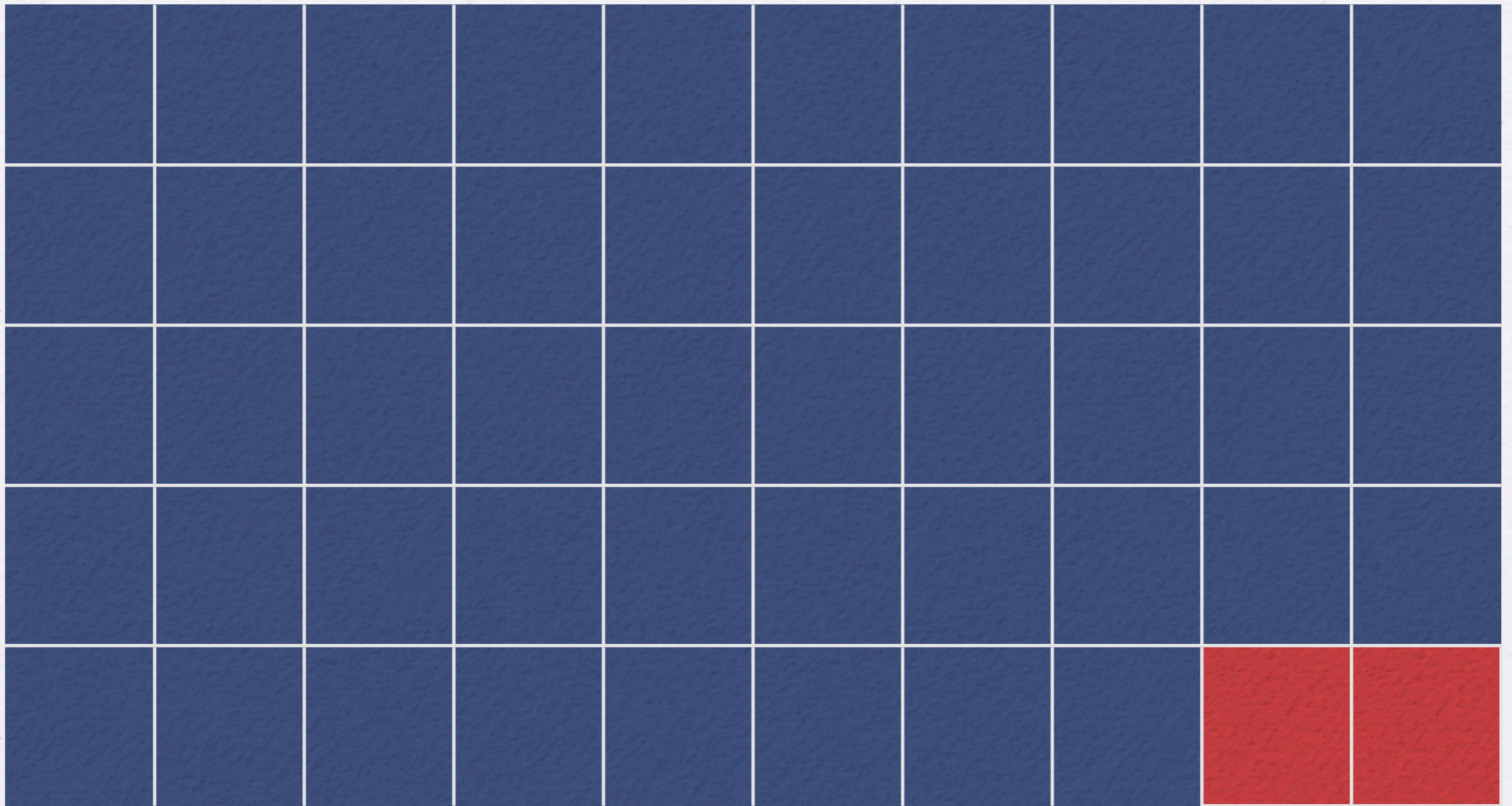
> look


**You see a tray containing a dish
(containing rice and peas) and
a fork.**



- 1. getFormattedList**
- 2. getSubtree / insertSubtree**
- 3. getTreeExplorer (read-only iterator)**
- 4. advance / enter / swapNodes**

value-based components



tree  list 

Game
World



Indexed
Tree



Map



Pointer

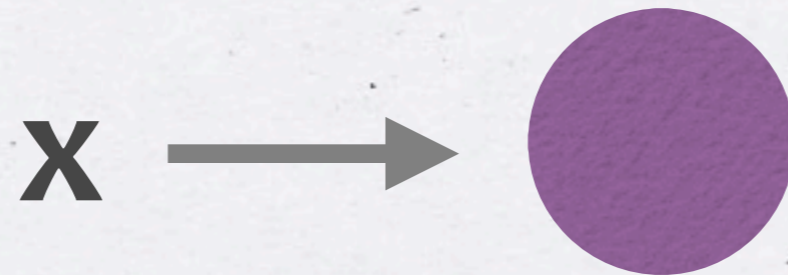
What did we learn?

- (1) Algorithms – similar to Java**
- (2) Data structures – difference**
 - (a) updates – swap in/out**
 - (b) strong ownership**

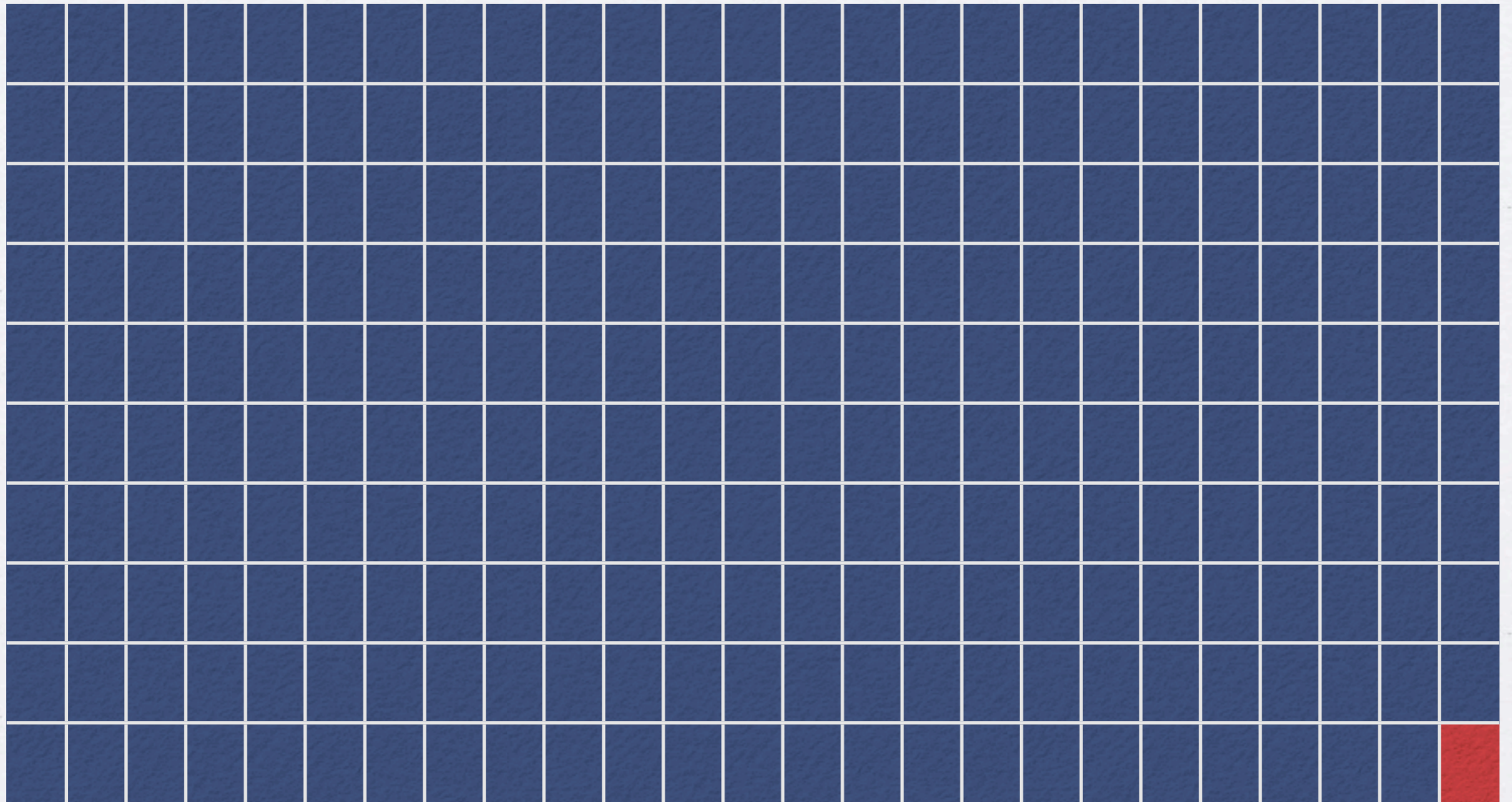
Questions?

alias avoidance

think “unique references”



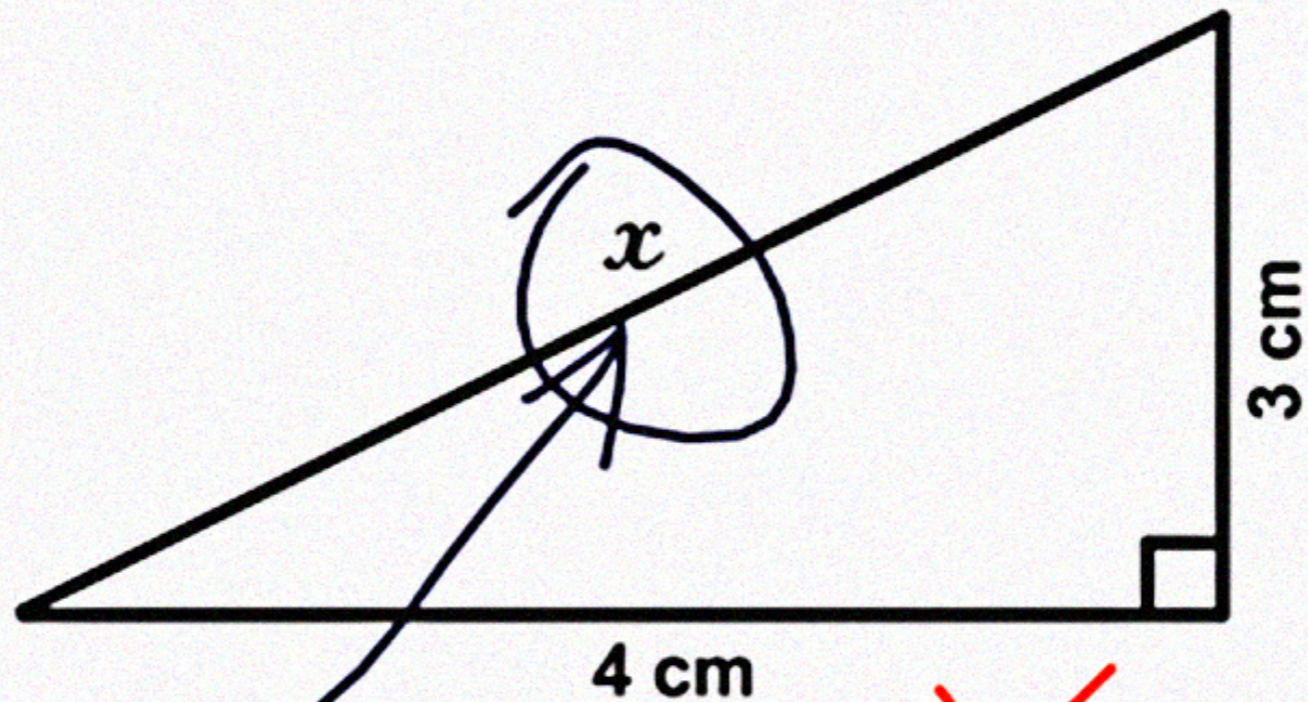
Obama-Palin



McCain-Biden



3. Find x .



Here it is



the simplest answer is not always correct