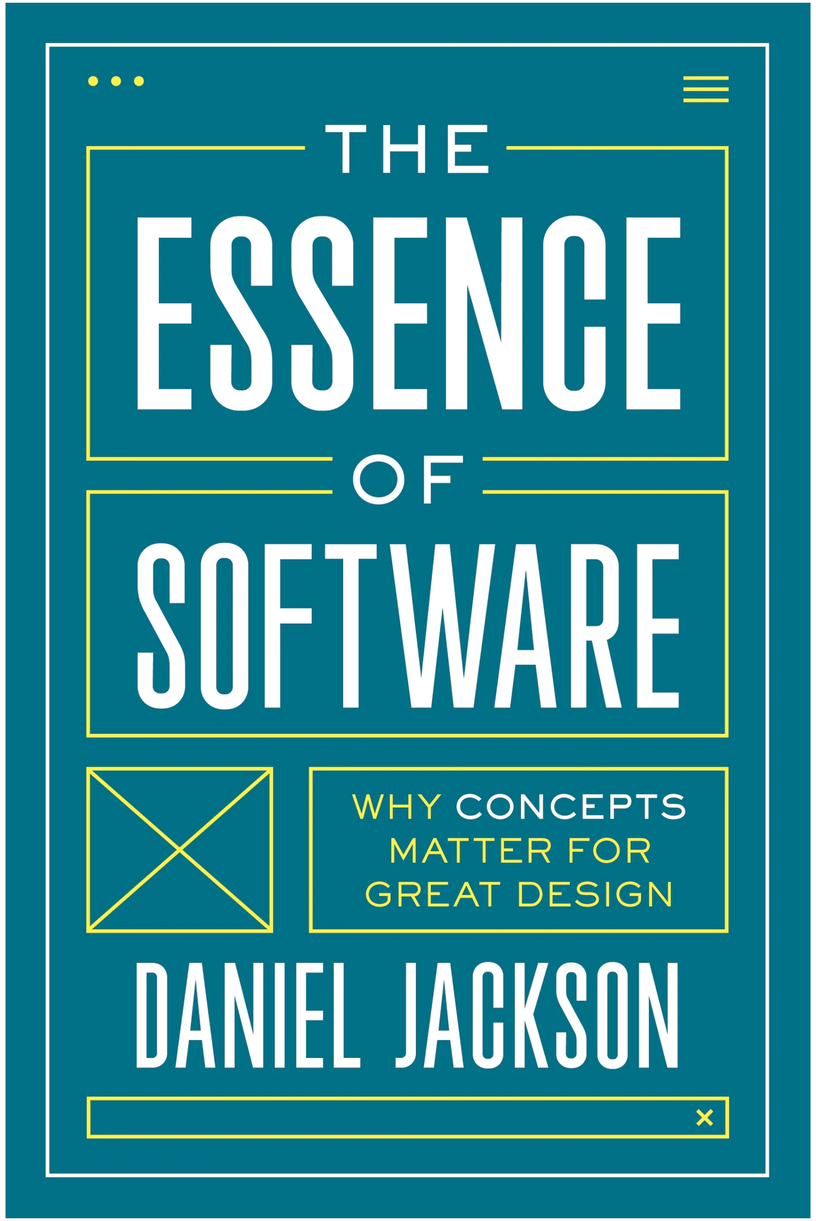
# Behavioral design with Alloy

# Software *concepts*

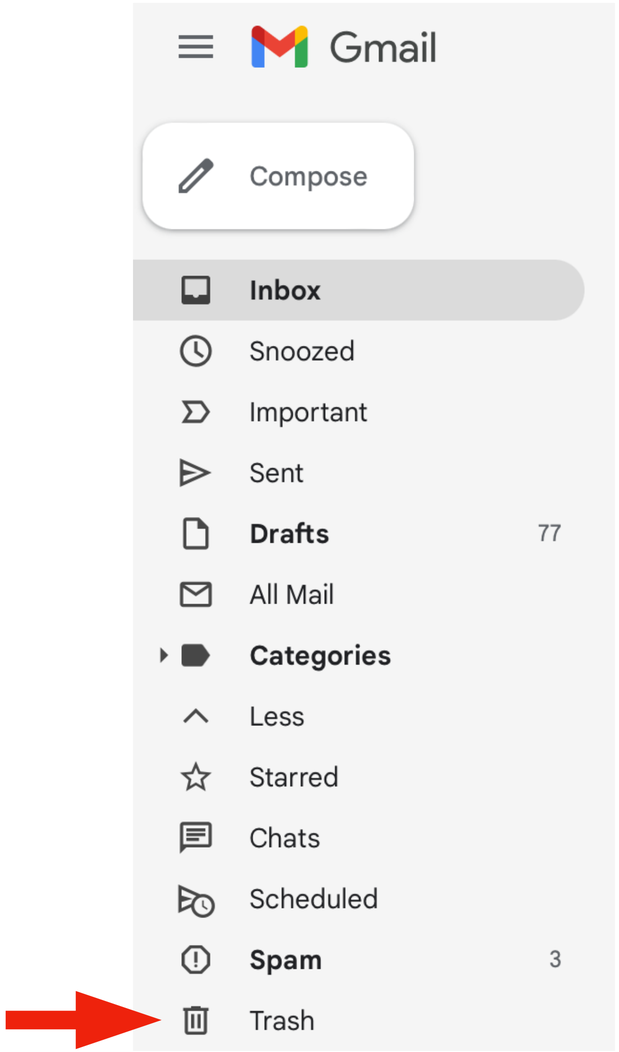

THE
ESSENCE
OF
SOFTWARE
WHY CONCEPTS MATTER FOR GREAT DESIGN
DANIEL JACKSON

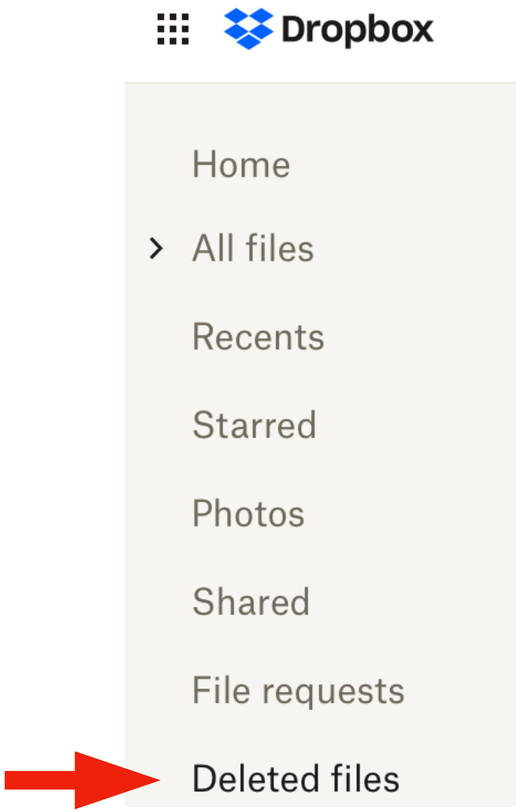
# *Concepts*

- Apps are made of recurring *concepts*
- Each *concept* is a self-contained unit of functionality with a clear purpose
- *Concepts* work together to provide the app overall functionality
- But can be understood independently of one another

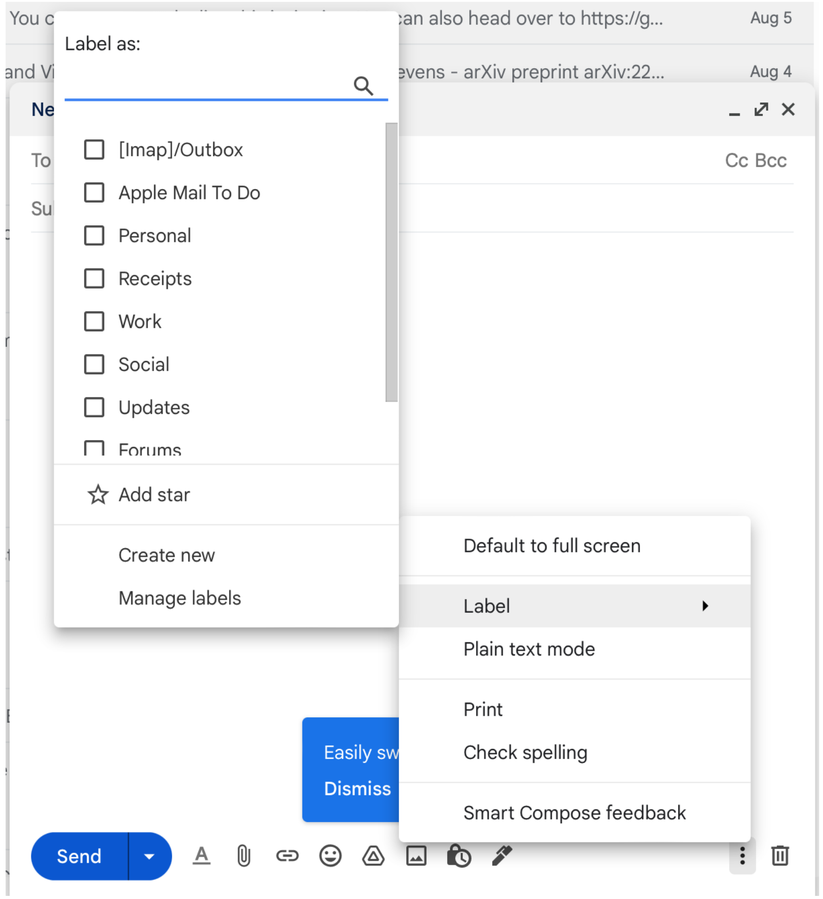
# Trash

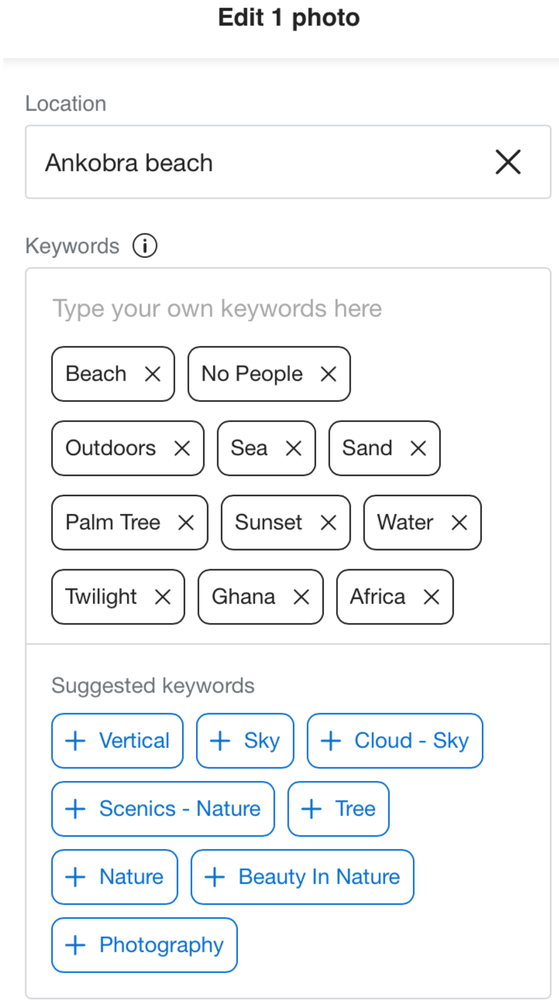

Open
Empty Trash

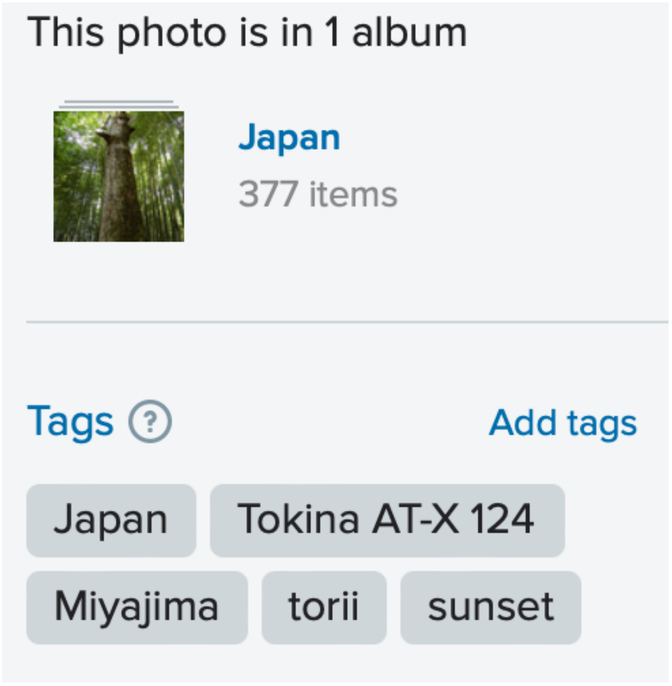

Gmail
Compose
Inbox
Snoozed
Important
Sent
Drafts 77
All Mail
Categories
Less
Starred
Chats
Scheduled
Spam 3
Trash


Dropbox
Home
All files
Recents
Starred
Photos
Shared
File requests
Deleted files

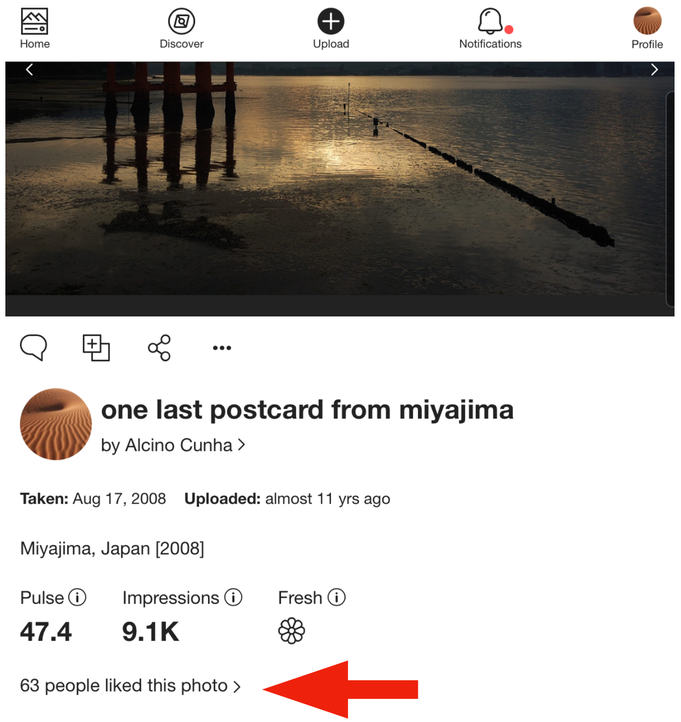
# Label

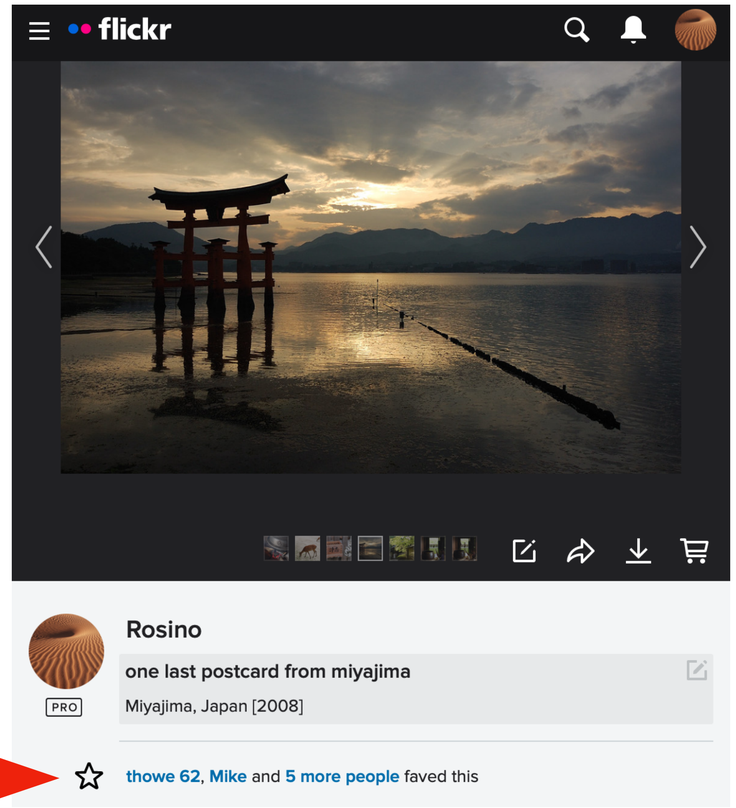

Label as:
[Imap]/Outbox
Apple Mail To Do
Personal
Receipts
Work
Social
Updates
Forums
Add star
Create new
Manage labels
Default to full screen
Label
Plain text mode
Print
Check spelling
Smart Compose feedback
Dismiss
Send
Aug 5
Aug 4
Cc Bcc

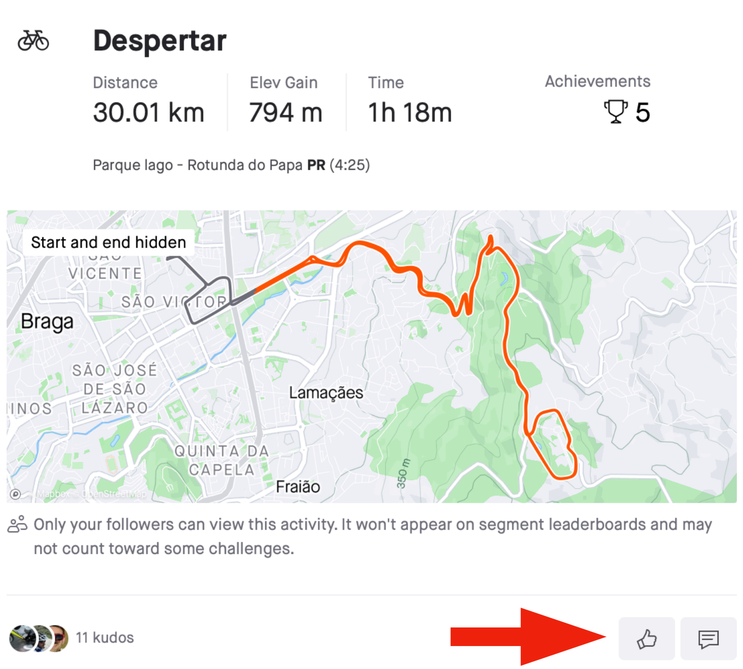

Edit 1 photo
Location
Ankobra beach
Keywords
Type your own keywords here
Beach
No People
Outdoors
Sea
Sand
Palm Tree
Sunset
Water
Twilight
Ghana
Africa
Suggested keywords
Vertical
Sky
Cloud - Sky
Scenics - Nature
Tree
Nature
Beauty In Nature
Photography


This photo is in 1 album
Japan
377 items
Tags
Add tags
Japan
Tokina AT-X 124
Miyajima
torii
sunset

# Like


Home
Discover
Upload
Notifications
Profile
one last postcard from miyajima
by Alcino Cunha
Taken: Aug 17, 2008 Uploaded: almost 11 yrs ago
Miyajima, Japan [2008]
Pulse
47.4
Impressions
9.1K
Fresh
63 people liked this photo


flickr
Rosino
PRO
one last postcard from miyajima
Miyajima, Japan [2008]
thowe 62, Mike and 5 more people faved this


Despertar
Distance
30.01 km
Elev Gain
794 m
Time
1h 18m
Achievements
5
Parque lago - Rotunda do Papa PR (4:25)
Start and end hidden
Braga
Lamaçães
Fraião
Only your followers can view this activity. It won't appear on segment leaderboards and may not count toward some challenges.
11 kudos

# *Concept* design

- Identify a clear purpose
- Choose the appropriate *state* and *actions* to fulfill that purpose
- The focus is on ensuring correctness and reusability

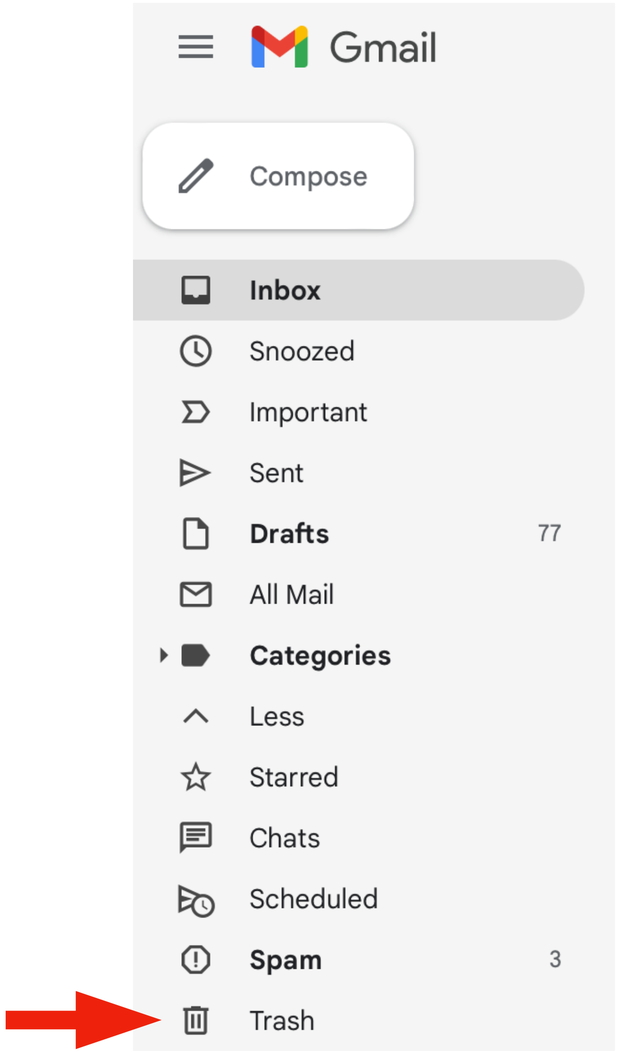
# App design

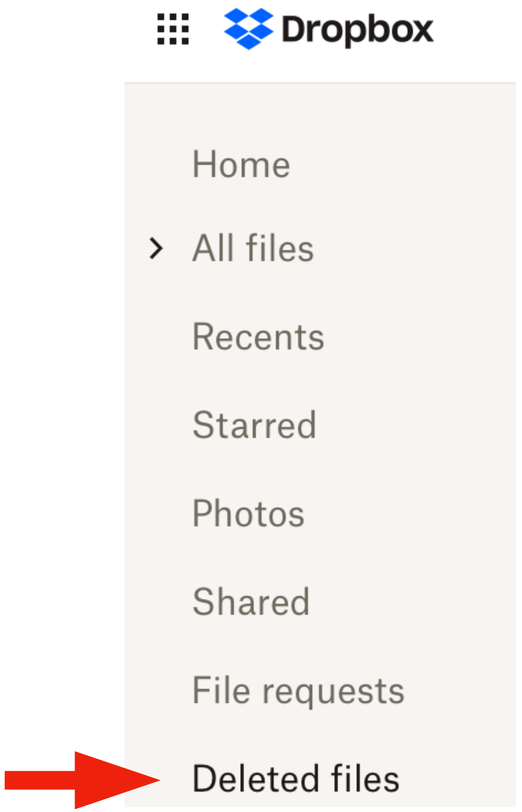
- Identify the core *concepts*
- Compose them, maybe providing new functionality
- The focus is on exploration

# Modeling *Concepts*

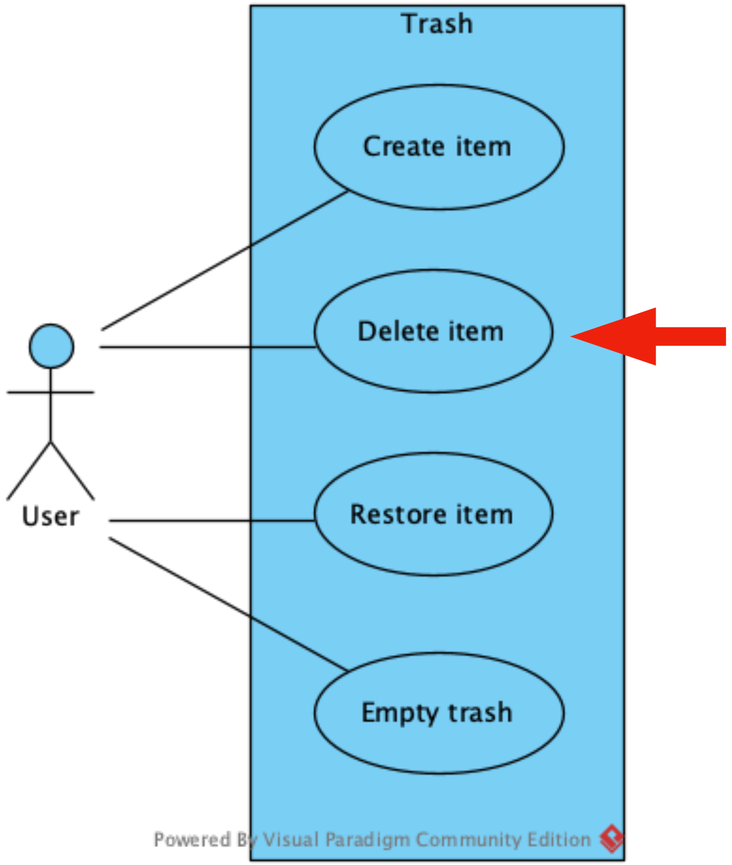
# Trash

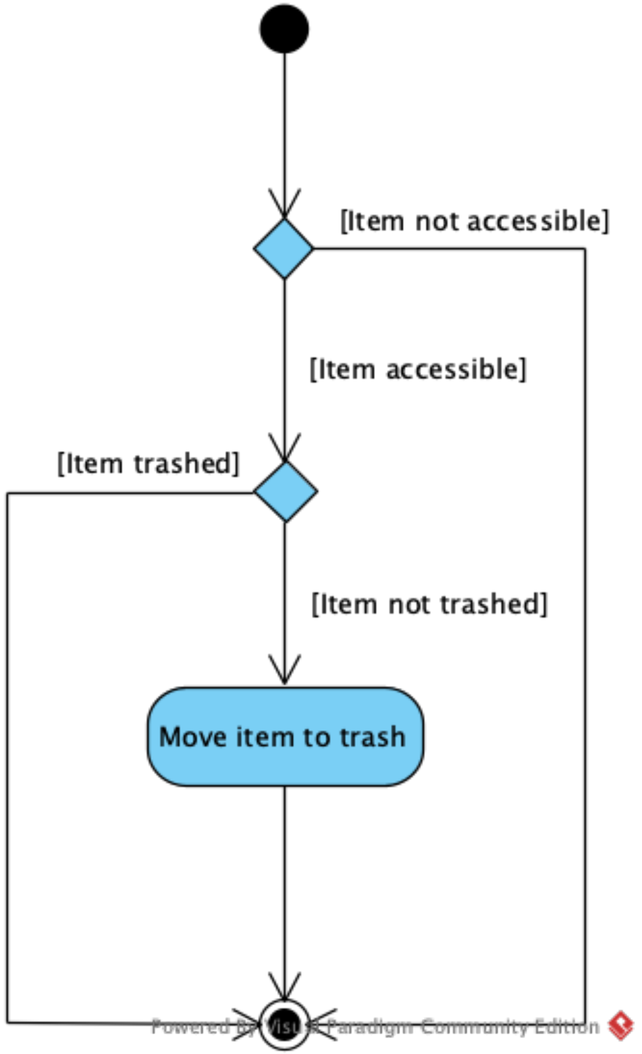

Open
Empty Trash


Gmail
Compose
Inbox
Snoozed
Important
Sent
Drafts 77
All Mail
Categories
Less
Starred
Chats
Scheduled
Spam 3
Trash


Dropbox
Home
All files
Recents
Starred
Photos
Shared
File requests
Deleted files

# Trash modeling *a la* UML



# Trash modeling *a la* Jackson

```
concept trash [Item]
purpose
  to allow undoing of deletions
state
  accessible, trashed : set Item
actions
  create (x : Item)
    when x not in accessible or trashed
    add x to accessible
  delete (x : Item)
    when x in accessible but not trashed
    move x from accessible to trashed
  restore (x : Item)
    when x in trashed
    move x from trashed to accessible
  empty ()
    when some item in trashed
    remove every item from trashed
operational principle
  after delete(x), can restore(x) and then x in accessible
  after delete(x), can empty() and then x not in accessible or trashed
```

# *Concept* modeling *a la* Jackson

- Name
  - Optionally parametrized by types that can be specialized when composing
- Purpose
  - A clear reason why you might want it
- State + Actions
  - A description of the concept behavior using a *transition system*
- Operational principle
  - Properties that show how the purpose is fulfilled by the actions

# Transition systems

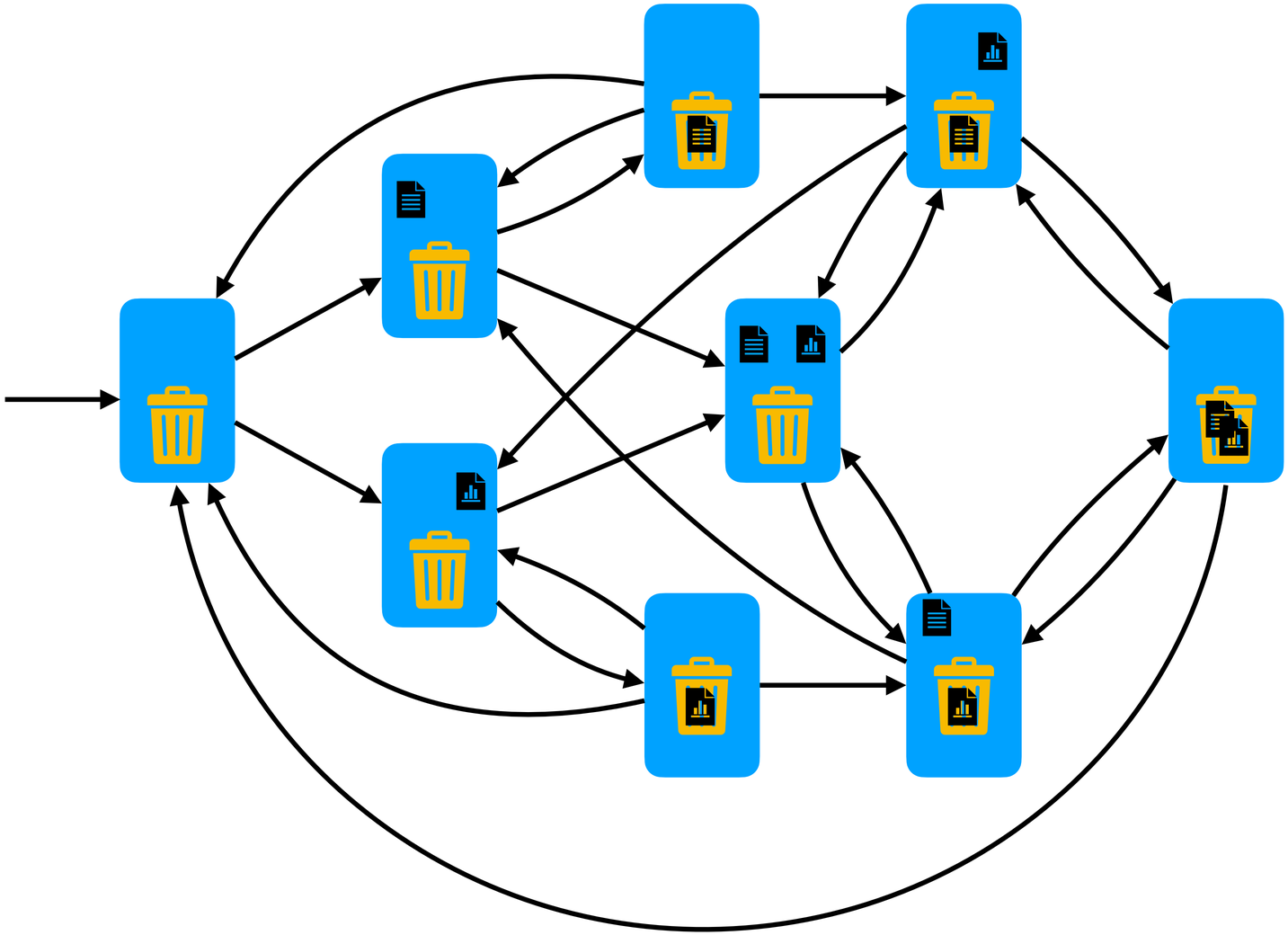
# Transition systems

- A popular model to describe the behavior of a system
- A *model* is often a synonym for a transition system
- There are many variants and related formalisms
  - Labeled transition systems
  - Kripke structures
  - Finite state machines
  - Hybrid and timed automata
  - ...

# States, transitions, and traces

- States
  - A *state* is a possible valuation to the structures of the system
  - *Initial* states describe how the system starts
- Transitions
  - A *transition* is a possible evolution between states
  - Transitions originate from actions of the system or the environment
- Traces
  - A *trace* is a sequence of states, describing a possible execution
  - A valid trace in a transition system is a path starting in an initial state

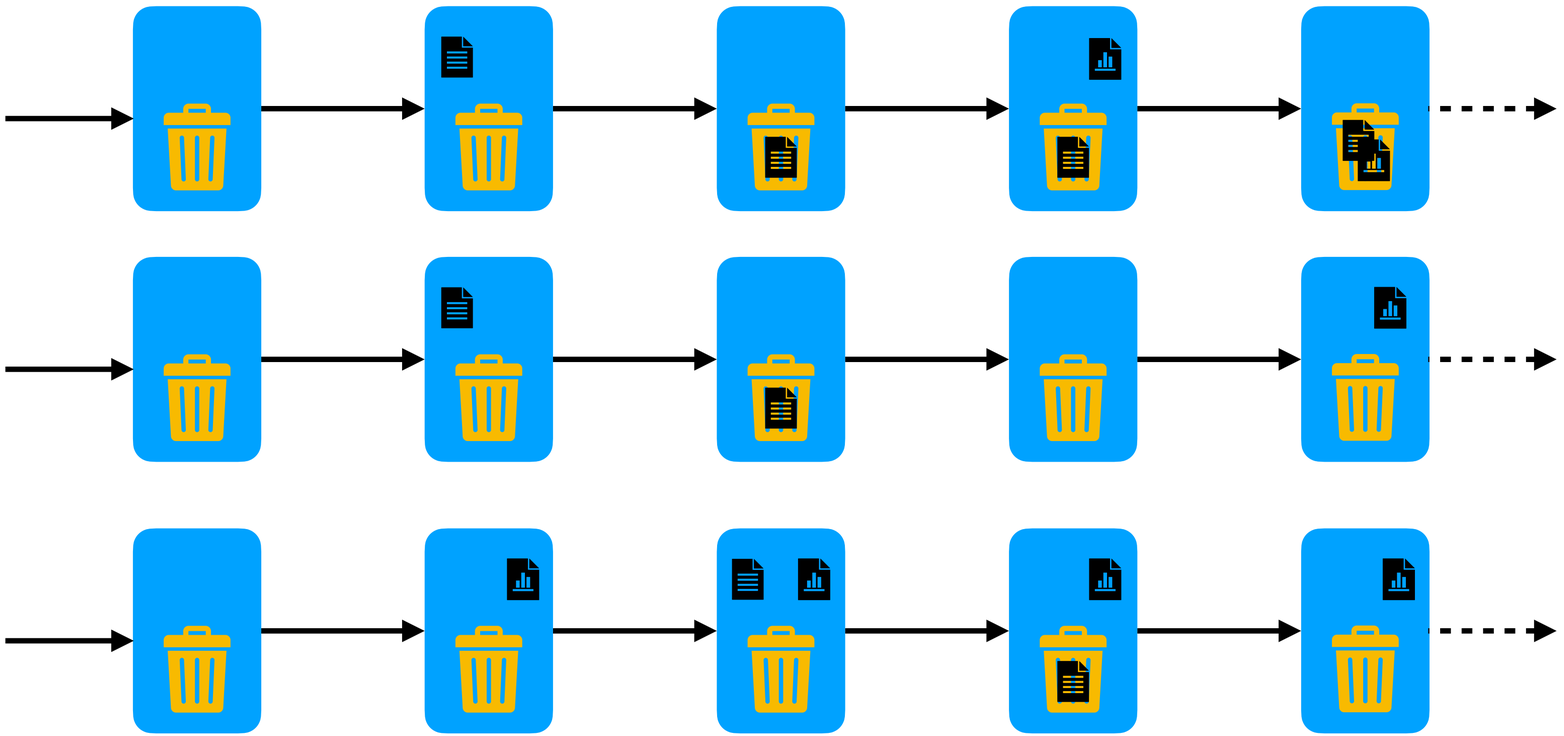
# Trash transition system



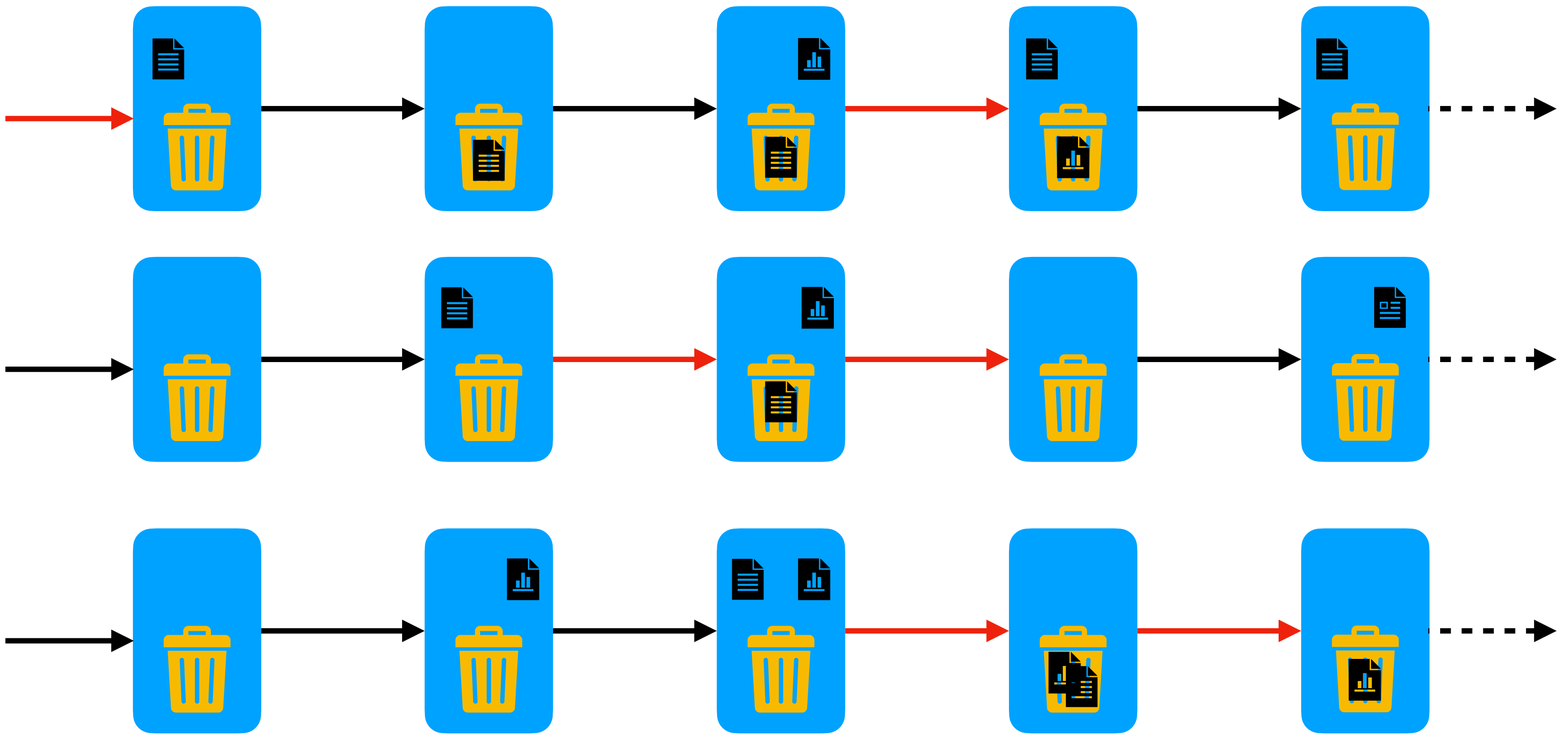
# Declarative modeling

- It is possible to describe a transition system by specifying instead which are its valid traces
- This requires specifying a property whose validity is established in a trace and not just in a single state
- The specification of properties about traces requires some sort of *temporal logic*

# Valid trash traces



# Invalid trash traces



# Specifying transition systems

# Mutability

- In Alloy 6 mutable signatures and fields can be declared with keyword **`var`**
  - Static field inside mutable signature yields a warning
  - Static signature extending or subset of a mutable one also yields a warning

# Trash states

```
sig Item {}
var sig Accessible in Item {}
var sig Trashed in Item {}
```

# Instances

- When mutable structures are declared, instances are **infinite** *traces*
- Analysis commands only return traces that can be represented finitely
  - Traces that loop back at some point
- Static signatures and fields are known as the *configuration* and have the same value in all states
- If there are mutable top-level signatures, **`univ`** (and **`iden`**) are also mutable

# Temporal logic

- Alloy 6 also supports linear *temporal logic*
- Temporal logic adds *temporal operators* to relational logic
- They allow us to “quantify” the validity of a formula over the different states of a trace
- A formula without temporal operators is only required to hold in the initial states
- Alloy 6 has both future and past temporal operators
- It also has the prime operator that denotes the value of a term in the next state

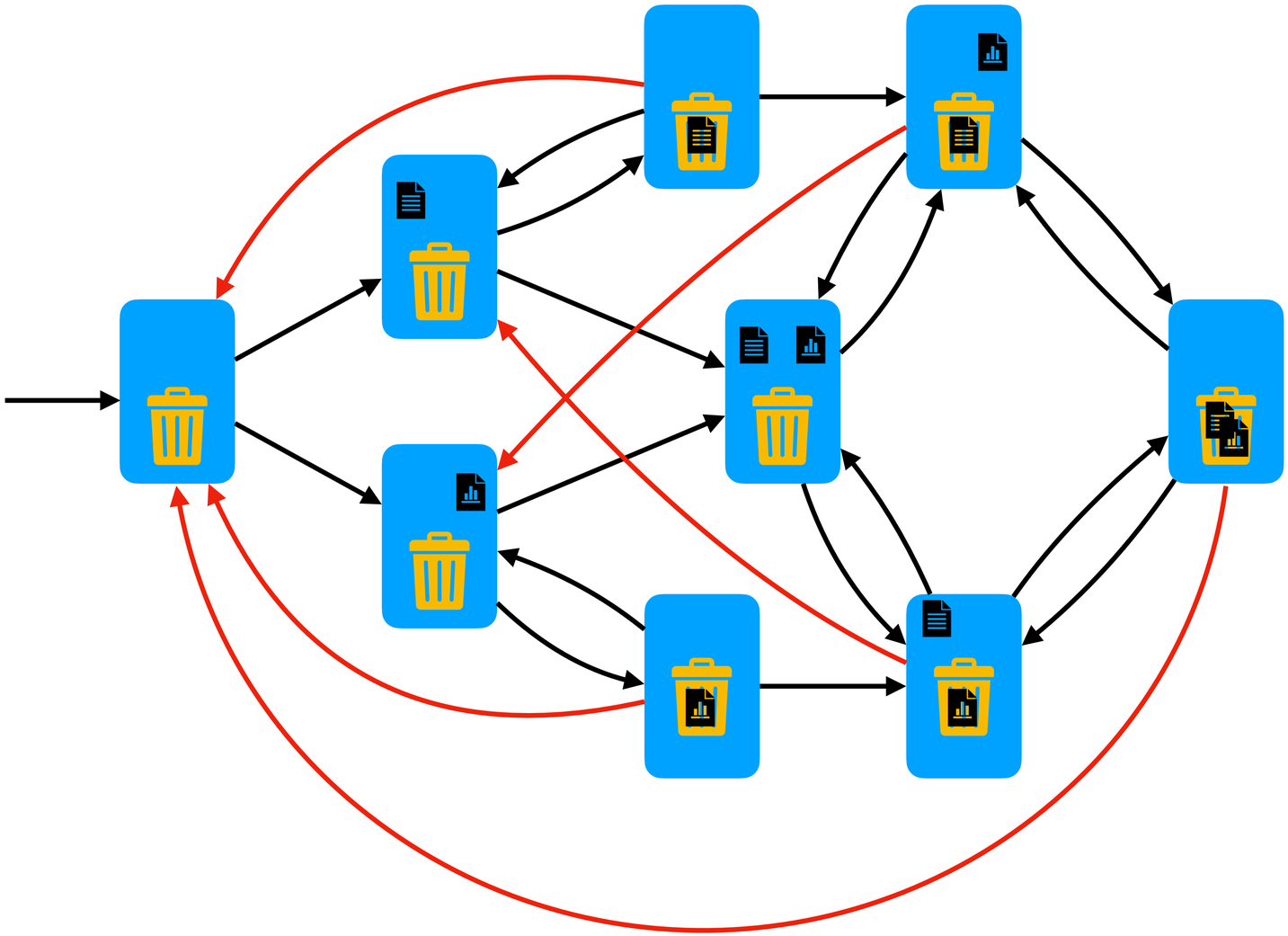
# Always, eventually, and prime

| | |
|---|---|
| **always** $\phi$ | $\phi$ is true in all future states |
| **eventually** $\phi$ | $\phi$ is true in some future state |
| $R'$ | The value of $R$ in the next state |

# Actions

- A set of transitions can be specified declaratively with an *action*
  - A formula without temporal operators, but including primed and unprimed variables
  - A condition without primed variables is a *guard* that specifies when is the action enabled
  - A condition with a primed variable is an *effect* that specifies what are the possible values for that variable after the action occurs
  - If a variable does not change, a *frame condition* should be included stating that the next value of the variable is the same
- By combining actions with the always temporal operator we can specify a system behavior
- Actions were first introduced by Leslie Lamport in the *Temporal Logic of Actions*

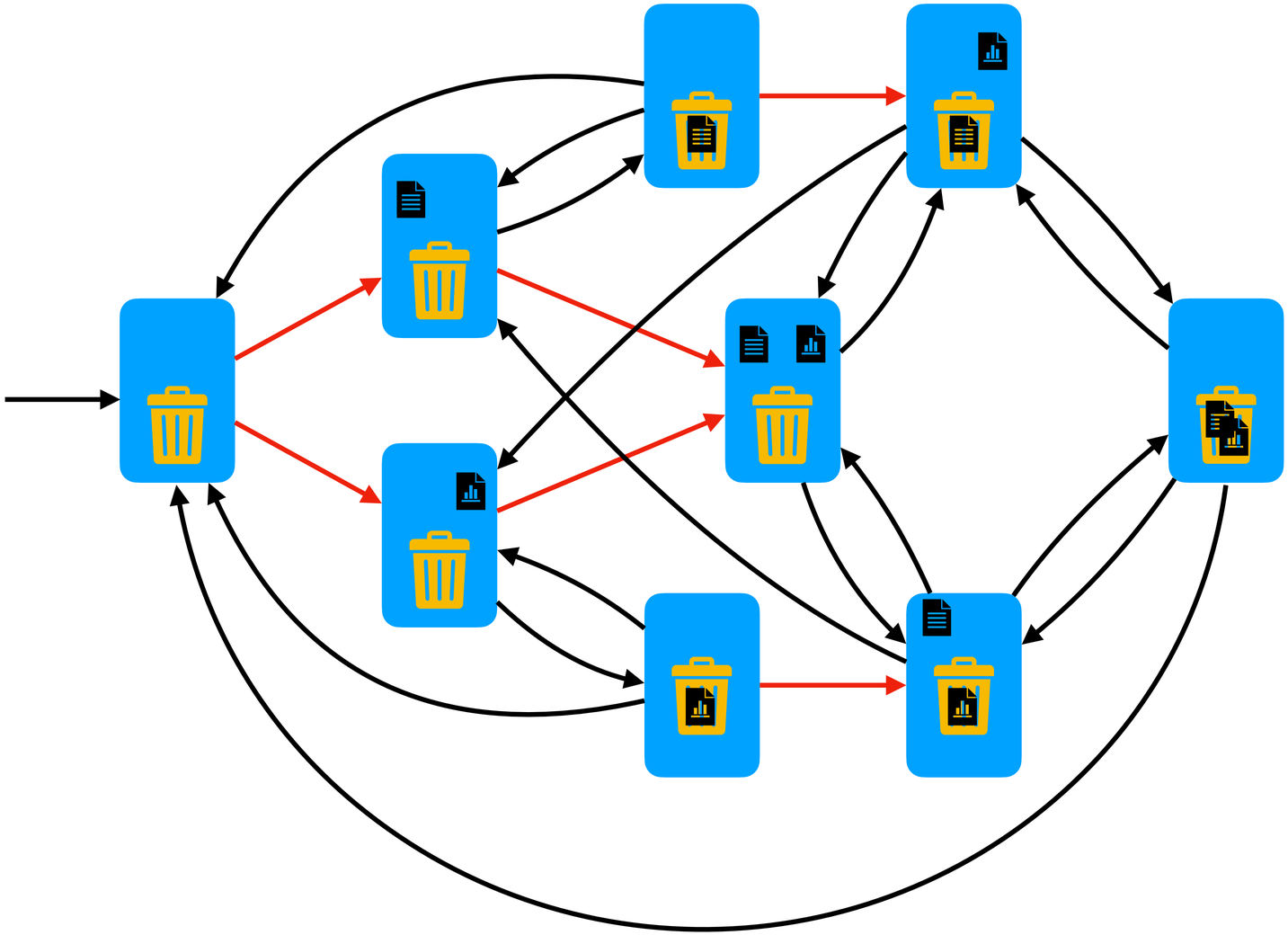
# Empty trash



# Empty trash

```
pred empty {
  // guard
  some Trashed
  // effect
  no Trashed'
  // frame condition
  Accessible' = Accessible
}
```

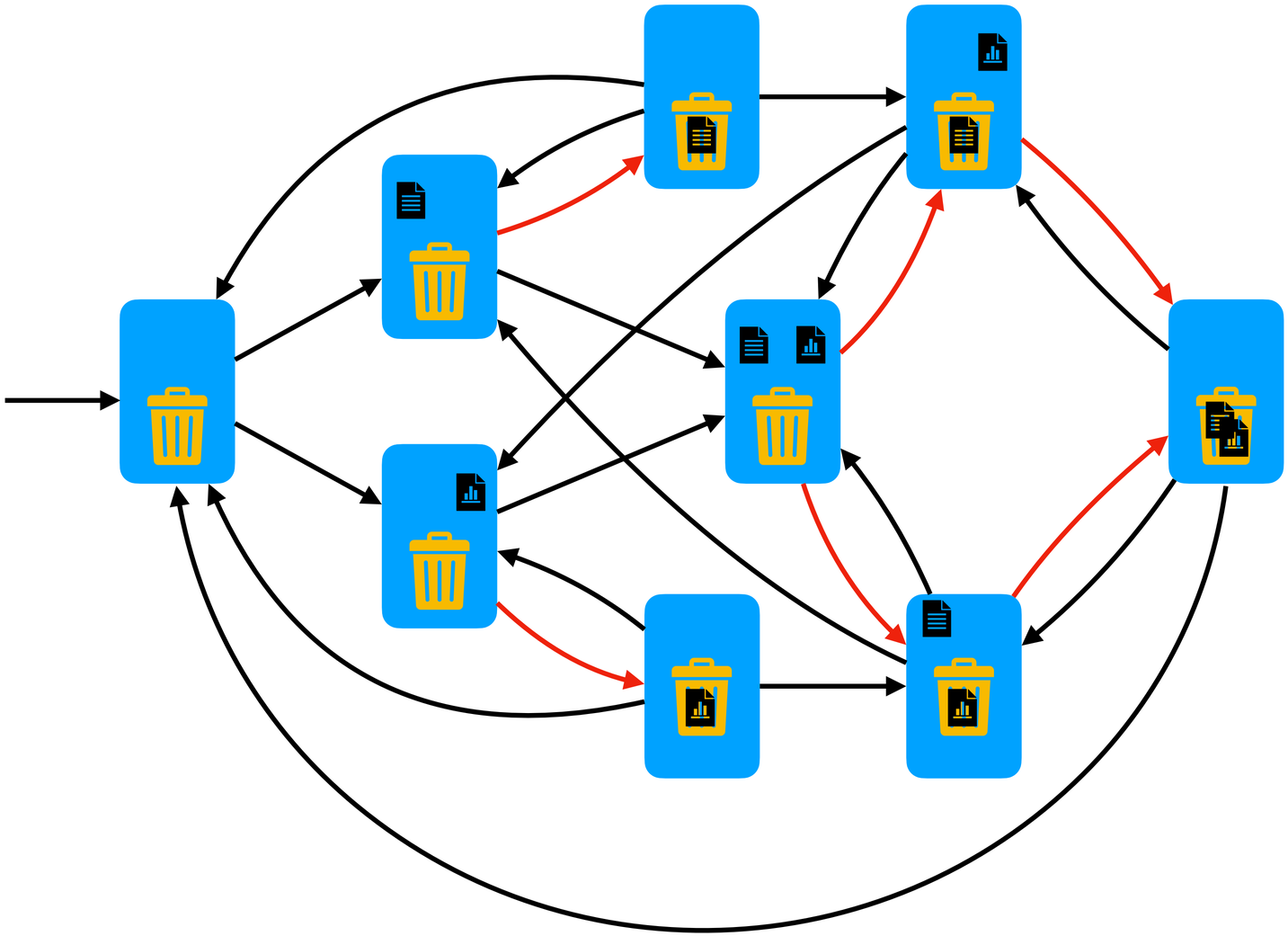
# Create item



# Create item

```
pred create [i : Item] {
  // guard
  i not in Accessible + Trashed
  // effect
  Accessible' = Accessible + i
  // frame condition
  Trashed' = Trashed
}
```

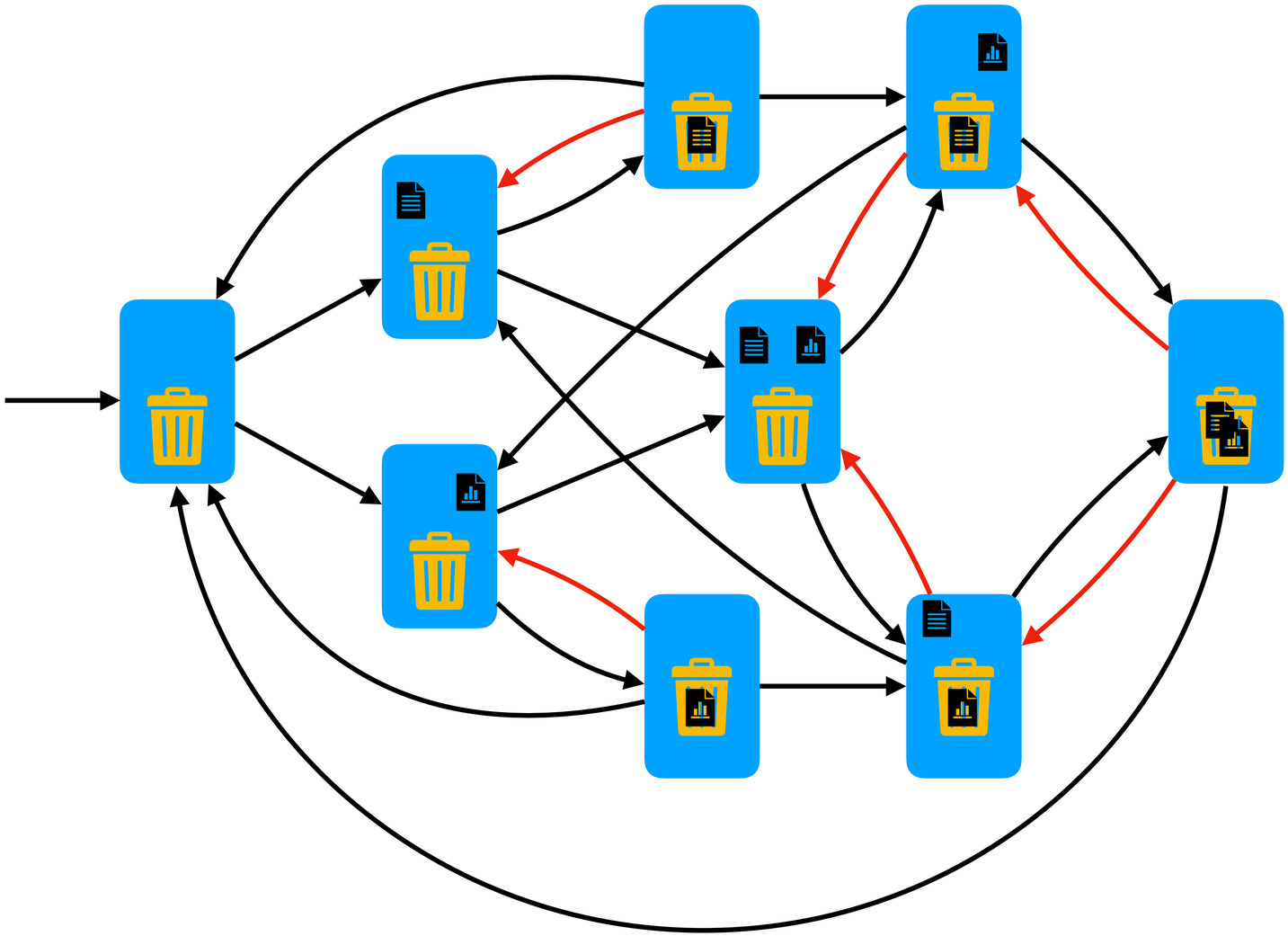
# Delete item



# Delete item

```
pred delete [i : Item] {
  // guard
  i in Accessible
  // effects
  Accessible' = Accessible - i
  Trashed' = Trashed + i
}
```

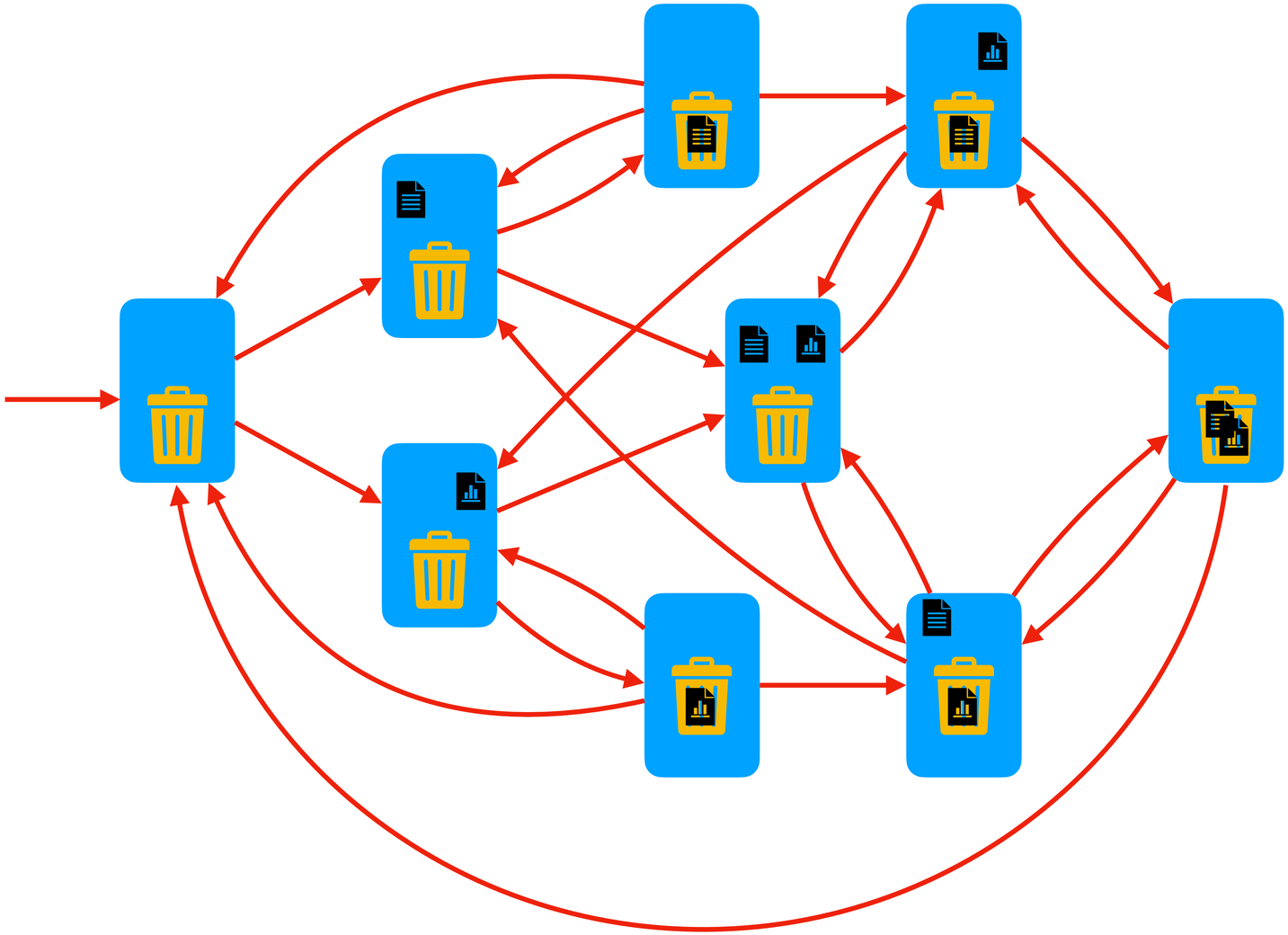
# Restore item



# Restore item

```
pred restore [i : Item] {
  // guard
  i in Trashed
  // effects
  Accessible' = Accessible + i
  Trashed' = Trashed - i
}
```

# Trash behavior



# Trash behavior

```
fact Behavior {
  // initial state
  no Accessible
  no Trashed
  // possible transitions
  always {
    (some i : Item | create[i] or delete[i] or restore[i])
    or
    empty
  }
}
```

# Validation

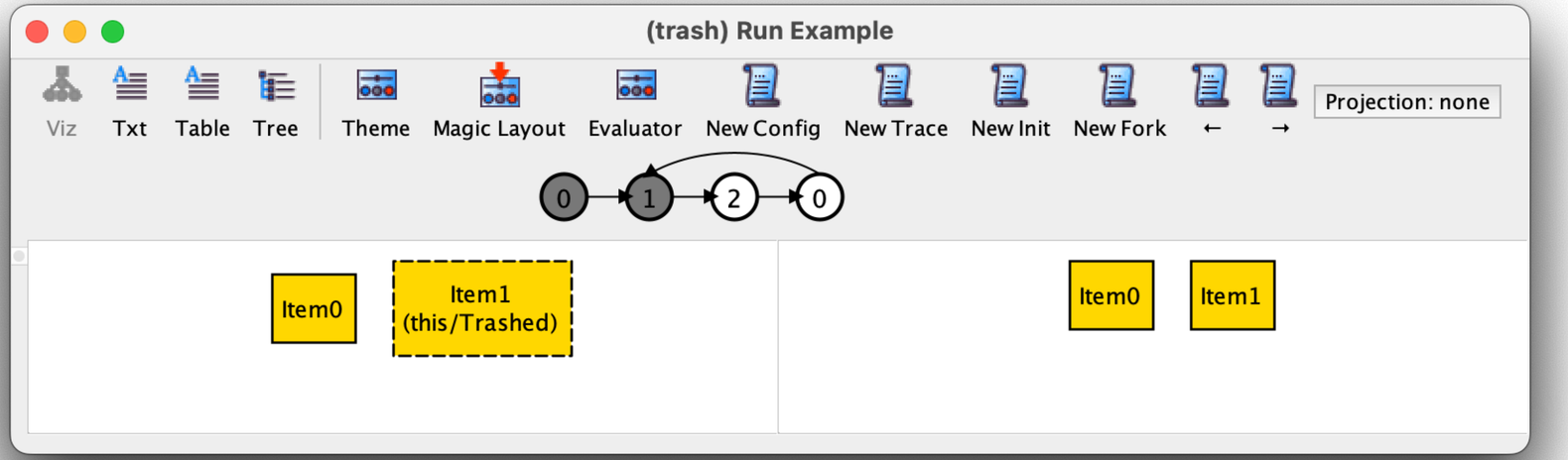
# Run commands

- As usual, **`run`** commands can be used to validate the model
- The scope of a mutable signature defines the maximum number of different atoms in the full trace, not a maximum per state

# Trace visualization

- The visualizer depicts two consecutive states of the trace side-by-side
  - By default mutable structures are depicted with dashed lines
- A representation of the infinite trace is shown above
  - Different states have different numbers
  - The loop back is explicitly depicted
  - Clicking on a state focus on that (and the succeeding) state
  - It is possible to move forwards and backwards in the trace with → and ←

# Trace visualization



# Simulation

- It possible to perform “simulation” with the *New* instance buttons
  - *New config*, returns a trace with a different configuration (a different value to the immutable structures)
  - *New trace*, returns any different trace with the same configuration
  - *New init*, returns a trace with the same config, but a different initial state
  - *New fork*, returns a trace with the same prefix, but a different next state

# Simulation



# Specifying scenarios

- A formula can be given in a **`run`** command to look for specific scenarios
- Keyword **`expect`** can be used to distinguish positive and negative scenarios

# Semi-colon

$\phi \;;\; \psi$ $\psi$ is valid after $\phi$

# Some trash scenarios

```
run Scenario1 {
  some i : Item {
    create[i]; delete[i]; restore[i]; delete[i]; empty
  }
} expect 1

run Scenario2 {
  some disj i,j : Item {
    create[i]; delete[j]
  }
} expect 0

run Scenario3 {
  some i : Item {
    create[i]; delete[i]; empty
  }
} for 1 Item expect 1
```

# Stuttering

# A clock specification

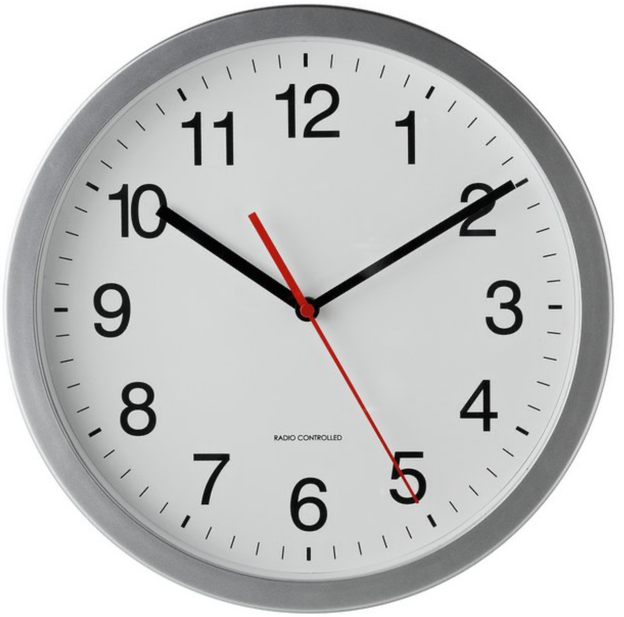
```
pred clock_spec {
   h = 12 and m = 0
   always {
      m'=(m+1)%60 and
      m=59 implies h'=(h%12)+1 else h'=h
  }
}
```



# Ceci n'est pas une montre?!

```
check clock_spec
```

```
Executing "Check clock_spec"
   Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
   1..2 steps. 55 vars. 12 primary vars. 59 clauses. 3ms.
   Counterexample found. Assertion is invalid. 3ms.
```



# A clock specification

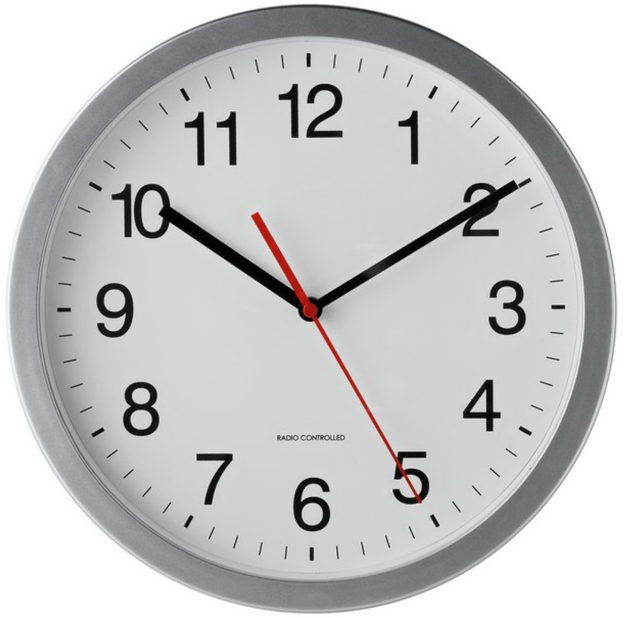
```
pred clock_spec {
   h = 12 and m = 0
   always {
      m'=(m+1)%60 and
      m=59 implies h'=(h%12)+1 else h'=h
      or
      m'=m and h'=h
  }
}
```



# Another clock

```
check clock_spec
```

```
Executing "Check clock_spec"
   Solver=sat4j Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
   1..10 steps. 151901 vars. 1875 primary vars. 413006 clauses. 1042ms.
   No counterexample found. Assertion may be valid. 298ms.
```



# Stuttering

- *Stuttering* can represent events by the environment or by other components of the system (not yet modeled)
- Stuttering enables *refinement*
  - adding detail or new components to a system
  - namely, it enables concepts to be composed to build apps
- In terminating systems, stuttering enables traces to be infinite

# Trash stuttering

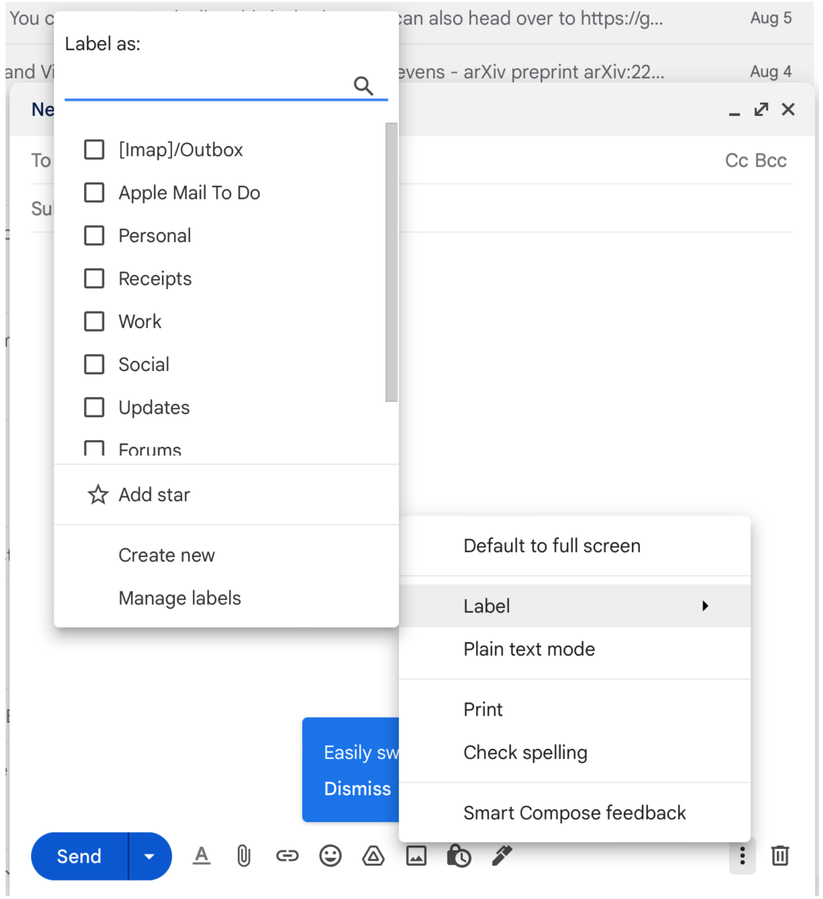
```
pred stutter {
   Accessible' = Accessible
   Trashed' = Trashed
}
```

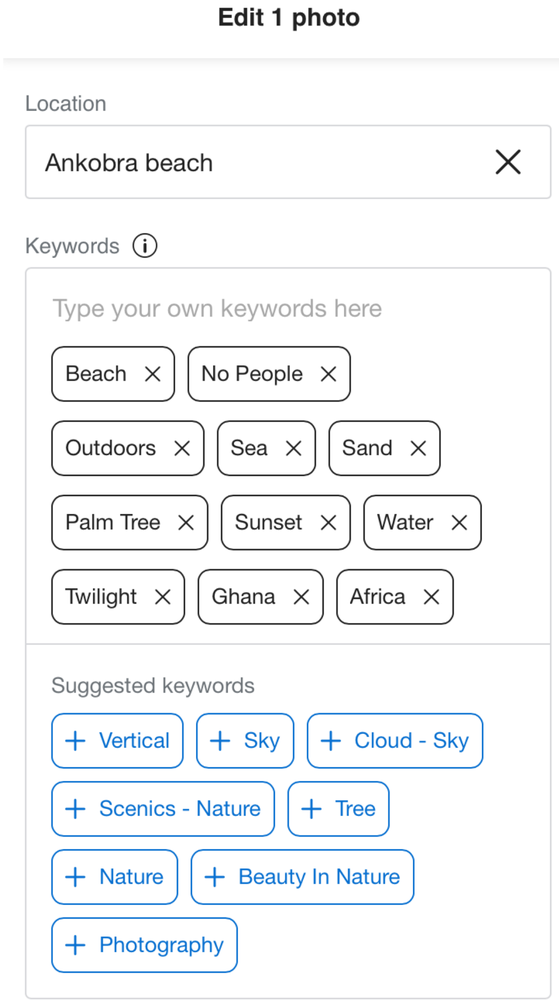
# Trash behavior

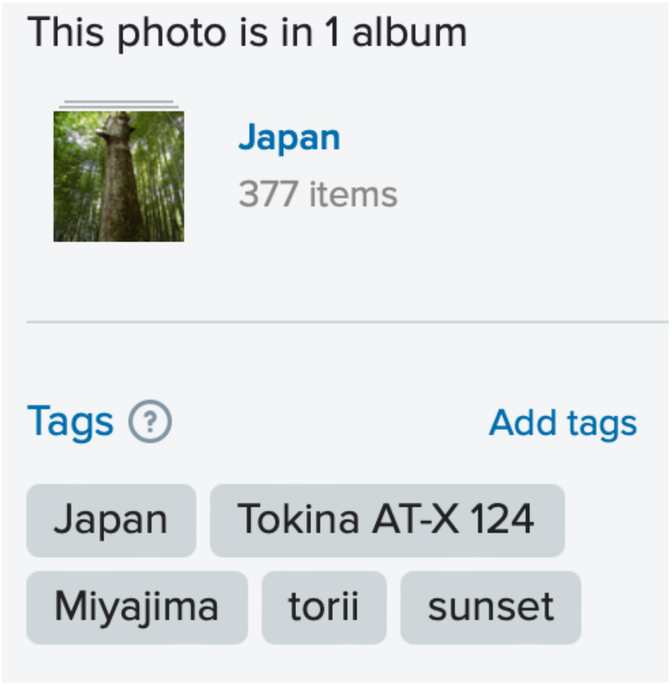
```
fact Behavior {
  // initial state
  no Accessible
  no Trashed
  // possible transitions
  always {
    (some i : Item | create[i] or delete[i] or restore[i])
    or
    empty
    or
    stutter
  }
}
```

# Another *concept*

# Label


Label as:
[Imap]/Outbox
Apple Mail To Do
Personal
Receipts
Work
Social
Updates
Forums
Add star
Create new
Manage labels
Default to full screen
Label
Plain text mode
Print
Check spelling
Smart Compose feedback
Dismiss
Send
Aug 5
Aug 4
Cc Bcc


Edit 1 photo
Location
Ankobra beach
Keywords
Type your own keywords here
Beach
No People
Outdoors
Sea
Sand
Palm Tree
Sunset
Water
Twilight
Ghana
Africa
Suggested keywords
Vertical
Sky
Cloud - Sky
Scenics - Nature
Tree
Nature
Beauty In Nature
Photography


This photo is in 1 album
Japan
377 items
Tags
Add tags
Japan
Tokina AT-X 124
Miyajima
torii
sunset

# Label modeling *a la* Jackson

```
concept label [Item]
purpose
  organize items into overlapping categories
state
  labels : Item -> set Label
actions
  affix (i : Item, l : Label)
    when l not in the labels of i
    add l to the labels of i
  detach (i : Item, l : Label)
    when l in the labels of i
    remove l from the labels of l
  clear (i : Item)
    when i has some labels
    remove all labels of i
operational principle
  after affix(i,l), while no detach(i,l) and no clear(i), i is in the labels of l
```

# The label in Alloy

```
sig Item {
  var labels : set Label
}
sig Label {}

fact Behavior {
  no labels
  always {
    (some i : Item, l : Label | affix[i,l] or detach[i,l])
    or
    (some i : Item | clear[i])
    or
    stutter
  }
}
```

# Affix label with *point-wise* effect

```
pred affix [i : Item, l : Label] {
  // guard
  l not in i.labels
  // effect
  i.labels' = i.labels + l
  // frame condition
  all j : Item - i | j.labels' = j.labels
}
```

# Affix label with *point-free* effect

```
pred affix [i : Item, l : Label] {
  // guard
  l not in i.labels
  // effect
  labels' = labels + i->l
}
```

# Detach label

```
pred detach [i : Item, l : Label] {
  // guard
  l in i.labels
  // effect
  labels' = labels - i->l
}
```

# Clear item

```
pred clear [i : Item] {
   // guard
   some i.labels
   // effect
   labels' = labels - i->Label
}
```

# Label scenarios

```
run Scenario1 {
  some i : Item, disj l,m : Label {
    affix[i,l]; affix[i,m]; clear[i]
  }
} expect 1

run Scenario2 {
  some i : Item, l : Label {
    affix[i,l]; affix[i,l]
  }
} expect 0
```