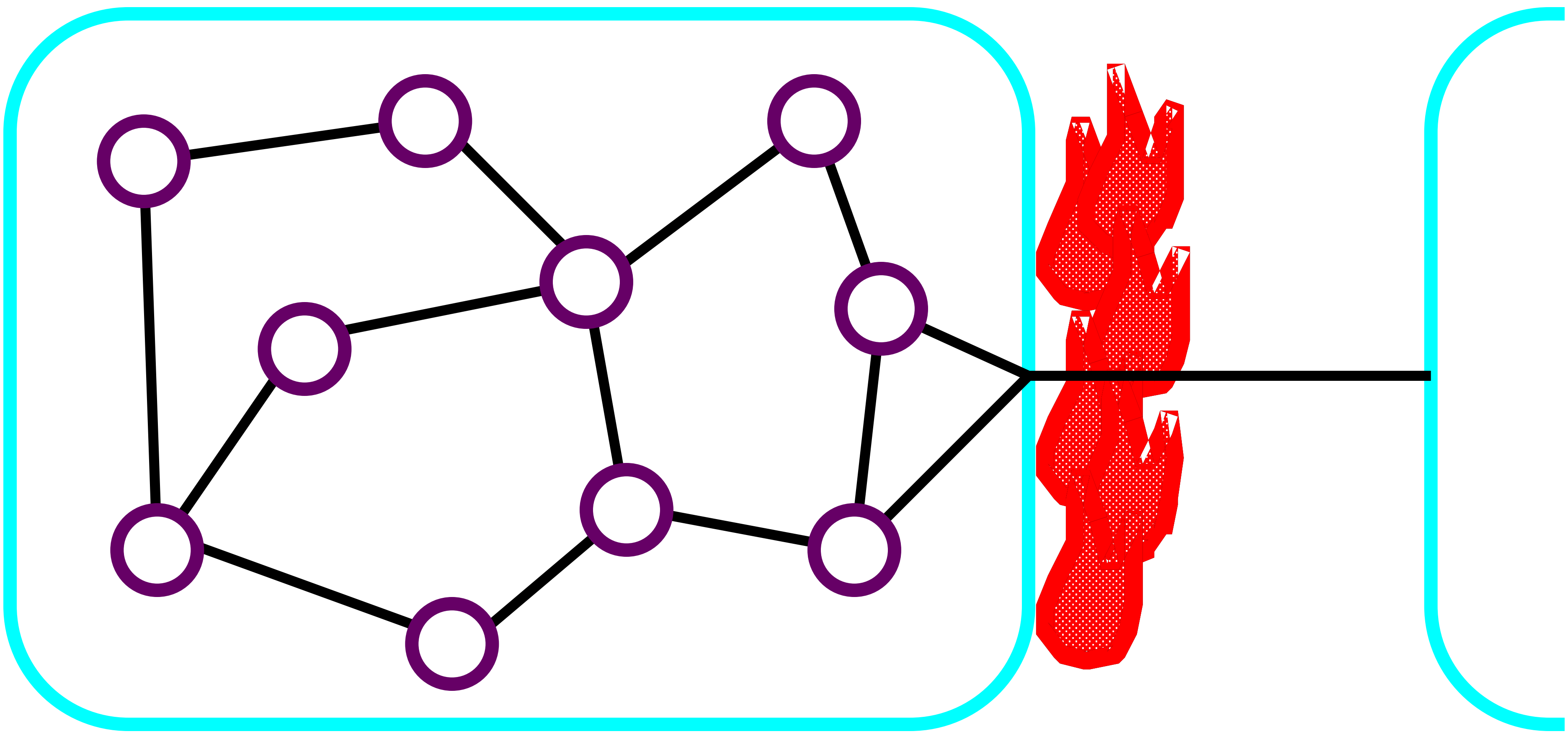
Wide Area Computation

Luca Cardelli

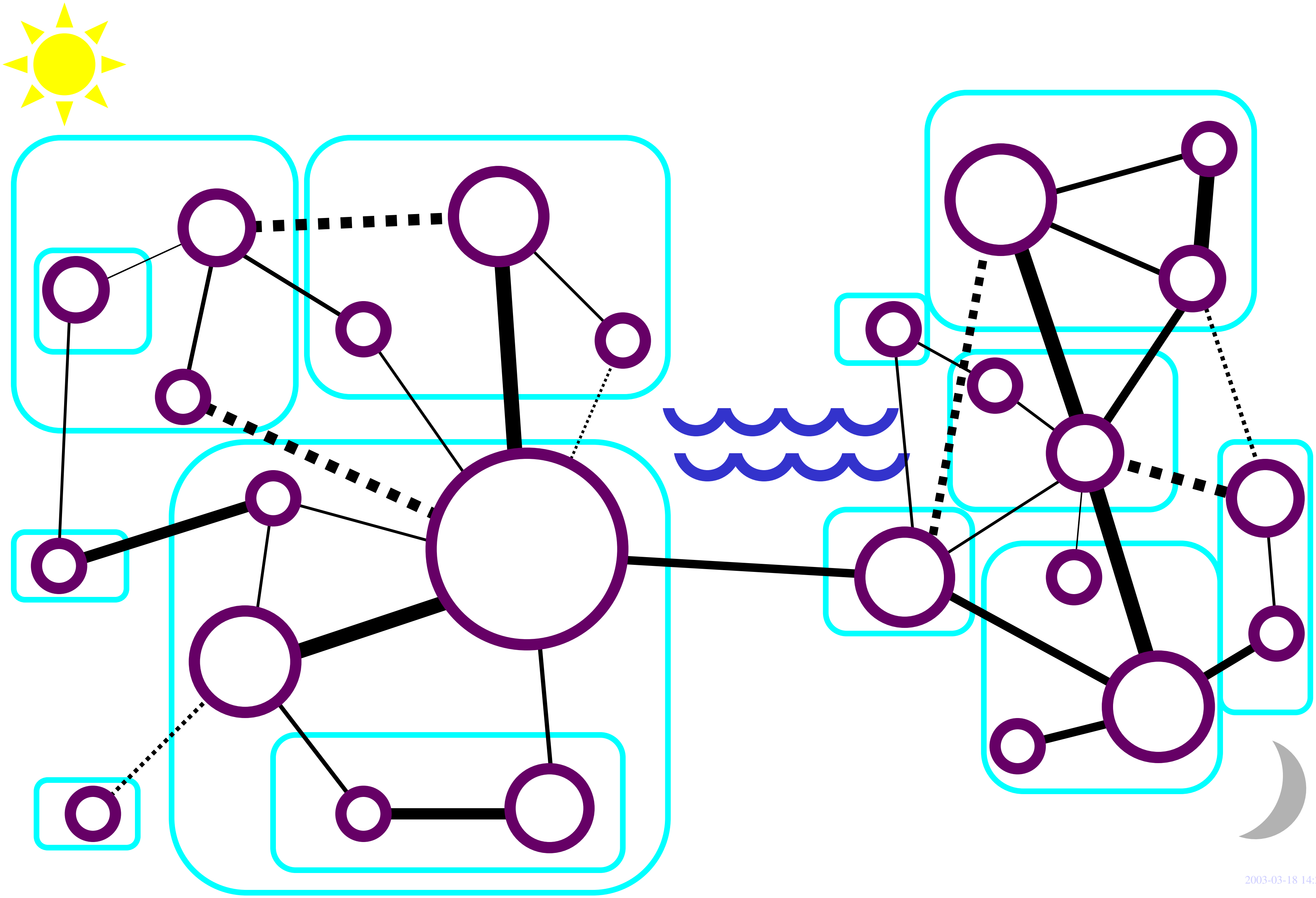
Microsoft Research

Valladolid 2000-11-10

LANs and (Traditional) Distributed Computing



Wide Area Networks

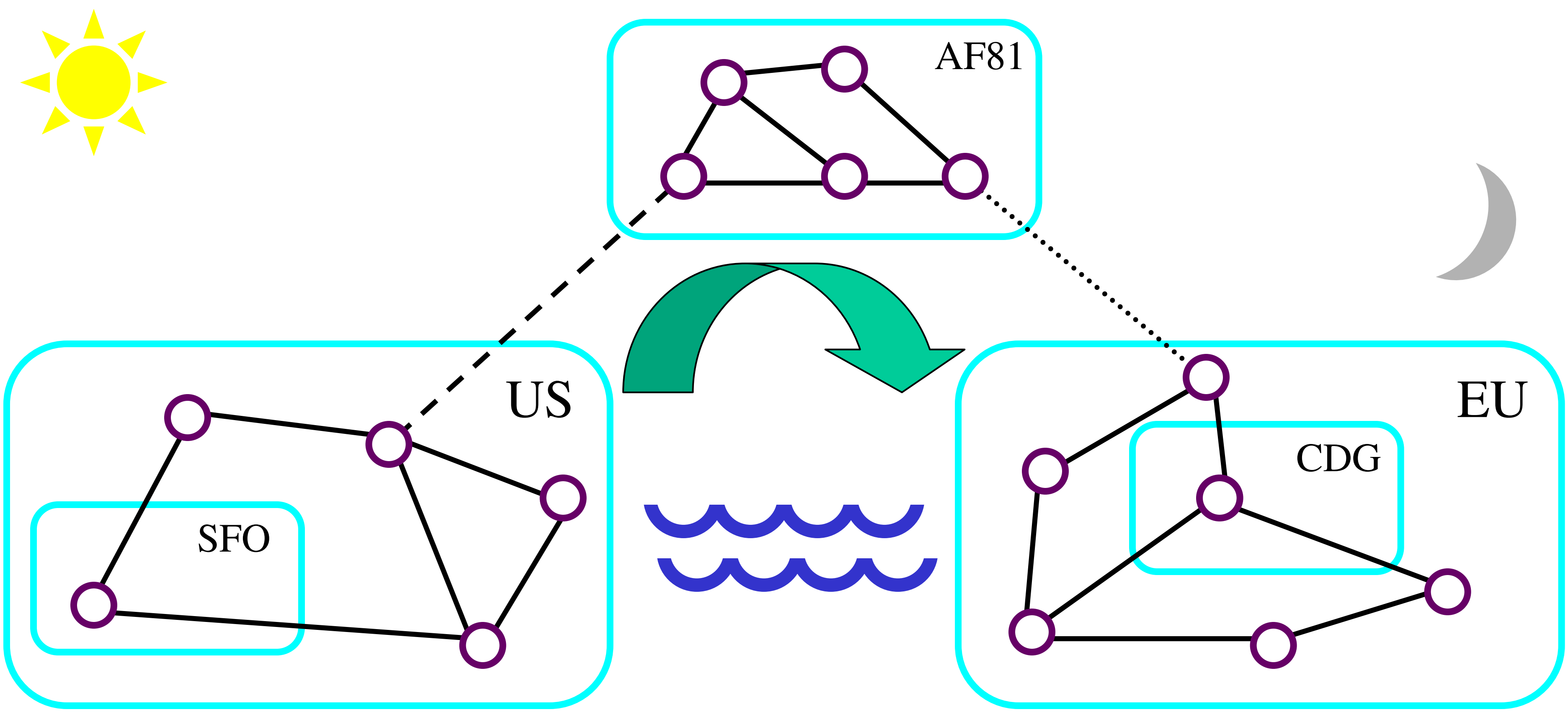


WAN Characteristics

- Internet/Web: a federated WAN infrastructure that spans the planet. We would like to program it.
- Unfortunately, federated WANs violate many familiar assumptions about the behavior of distributed systems.
- Three phenomena that remain largely hidden in LANs become readily observable:
  - Virtual locations.
  - Physical locations.
  - Bandwidth fluctuations.
- Another phenomenon becomes unobservable:
  - Failures.

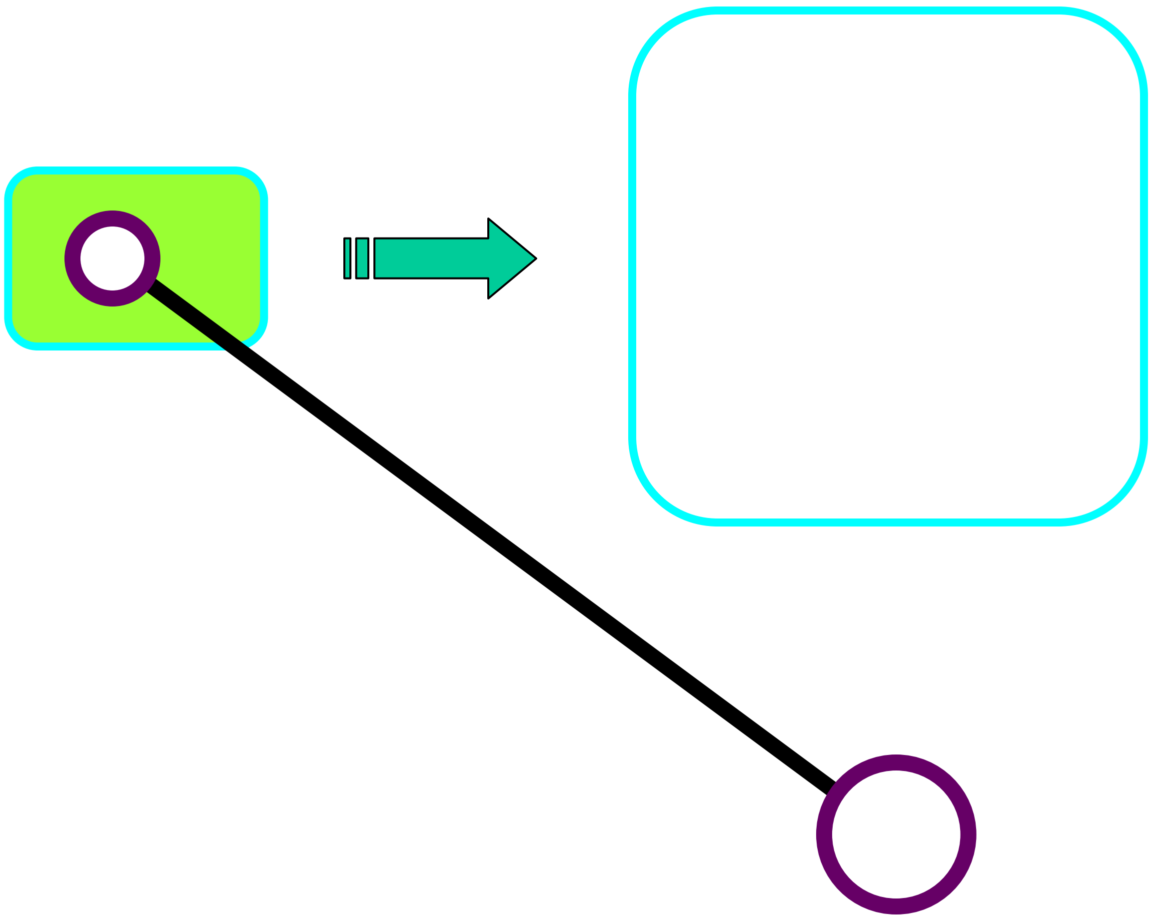
Mobile Computing

- (Software) Active computations move around.
- (Hardware) Mobile devices transport active computations.



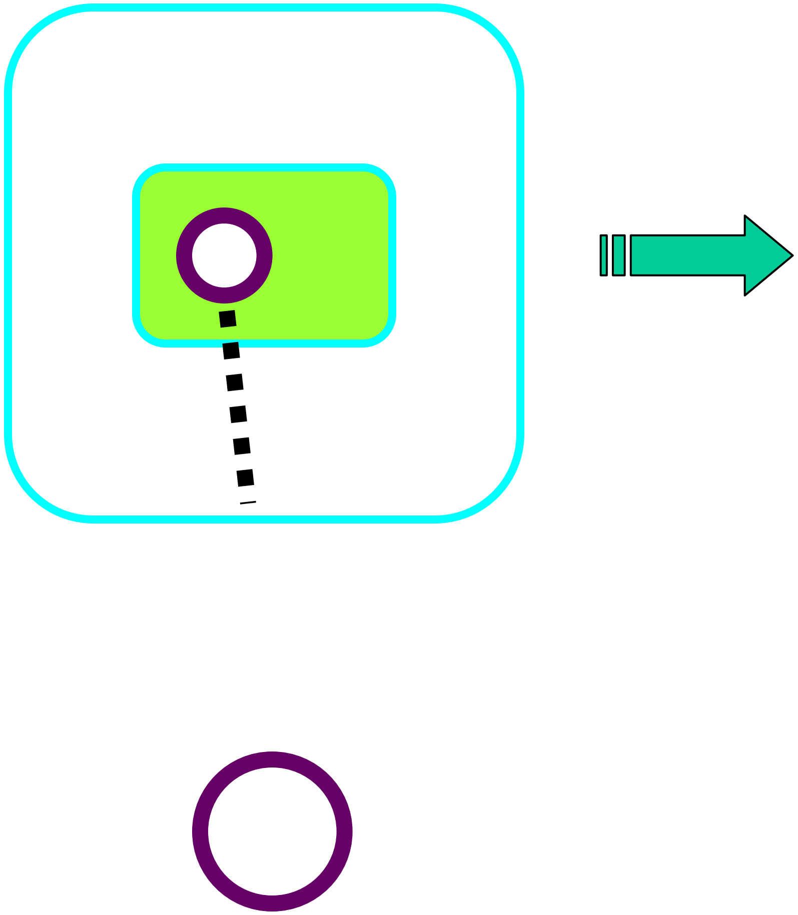
Connectivity Depends on Location

- Tunneling.
  - Accidental disconnection (bad infrastructure, solar flares).
  - Intentional disconnection (privacy, security, quiet).



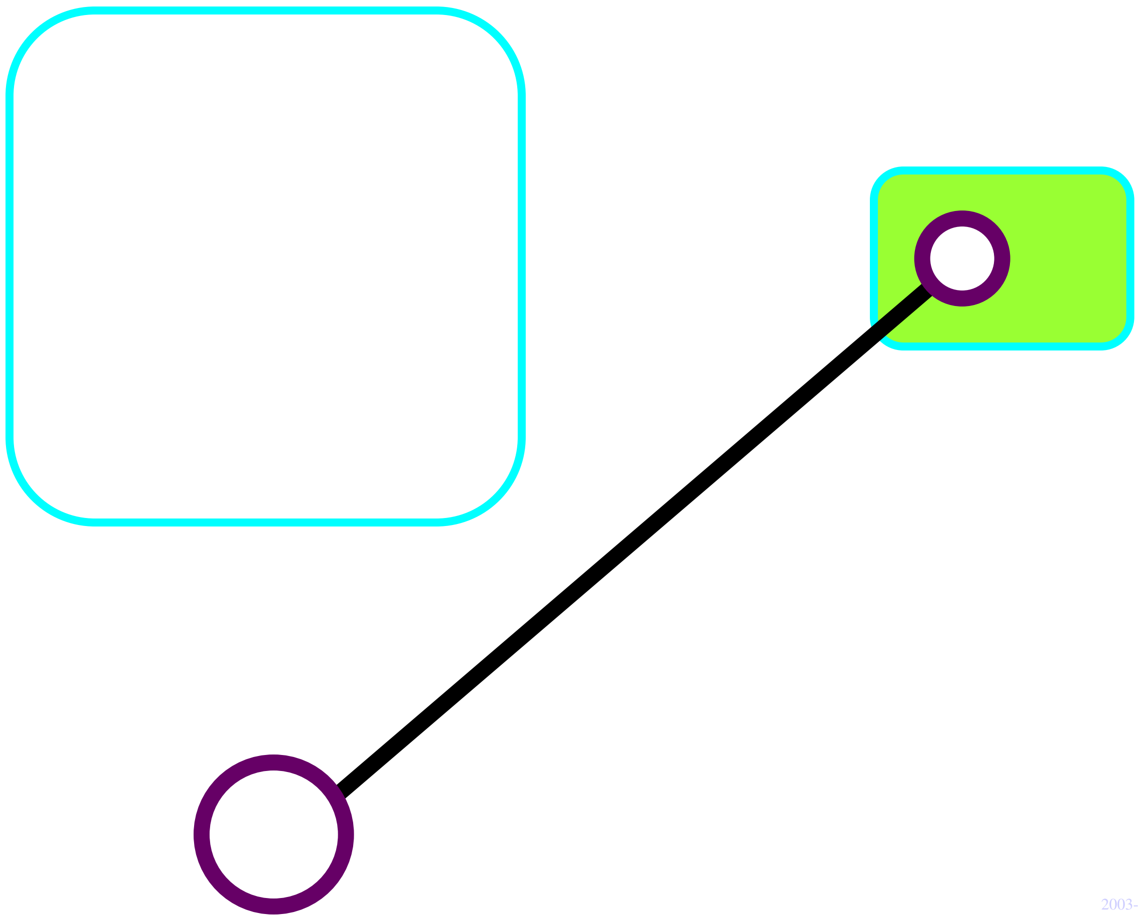
Connectivity Depends on Location

- Tunneling.
  - Accidental disconnection (bad infrastructure, solar flares).
  - Intentional disconnection (privacy, security, quiet).



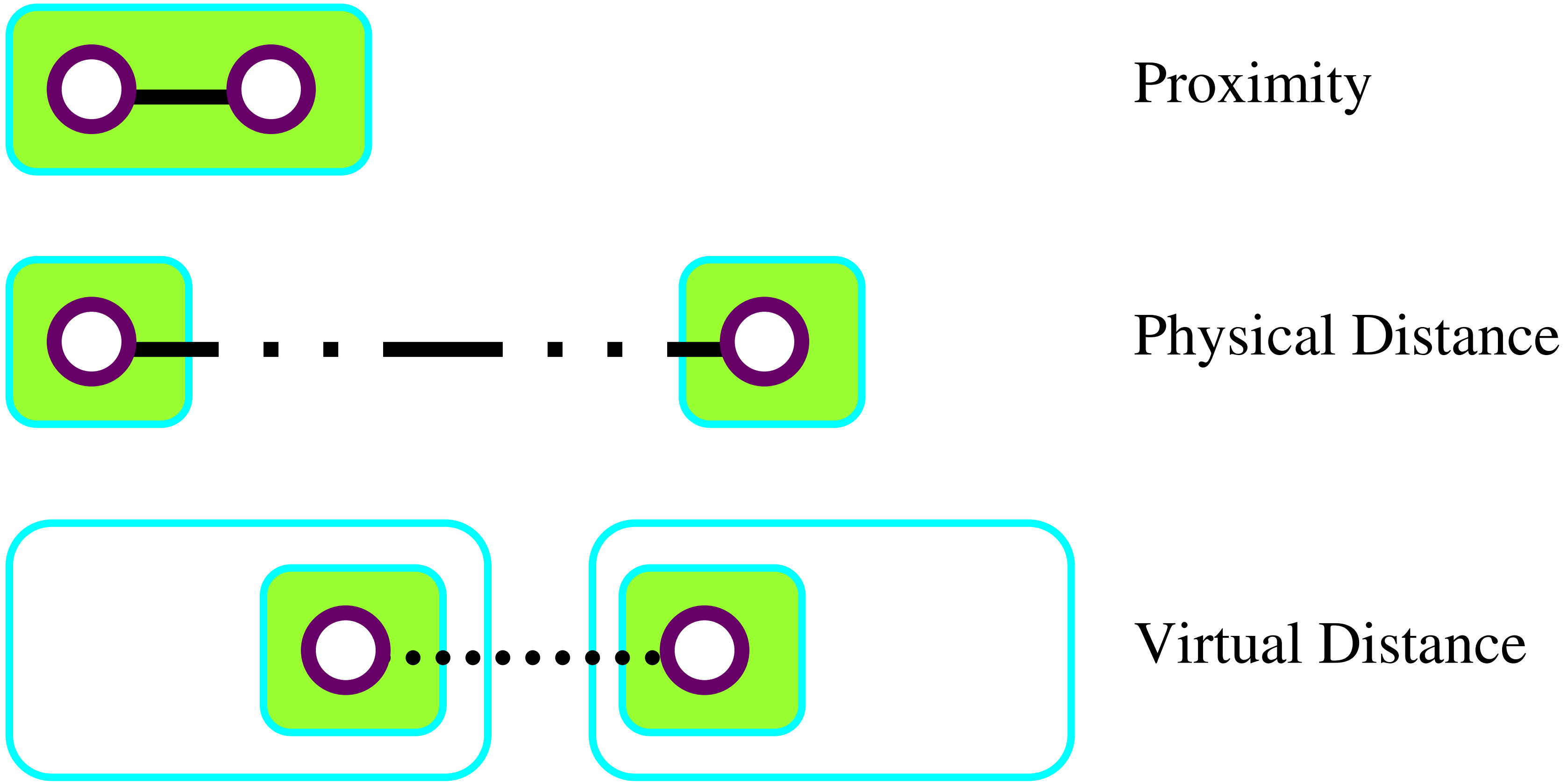
Connectivity Depends on Location

- Tunneling.
  - Accidental disconnection (bad infrastructure, solar flares).
  - Intentional disconnection (privacy, security, quiet).



Connectivity Depends on Proximity

- No remote real-time control.
- No secure (unencrypted) remote links.



Related Work

- Broadly classifiable in two categories:
  - Agents (Actors, Process Calculi, Telescript, etc.)
  - Spaces (Linda, Distributed Lindas, JavaSpace, etc.)
- With our work on Ambients, we aim to unify and extend those basic concepts.

Modeling Mobility: Barriers

- Locality: Barrier topology.
  - Cf. failure semantics, named processes.
- Process mobility: Barrier crossing.
  - Cf. name passing (π), process passing (CHOCS).
- Security: (In)ability to cross barriers.
  - Cf. cryptography (Spi), flow control (SLAM)
- Interaction: Shared ether within a barrier.
  - No action at a distance. No global channels. No preservation of connectivity (channels, tethers) across barriers.

Approach

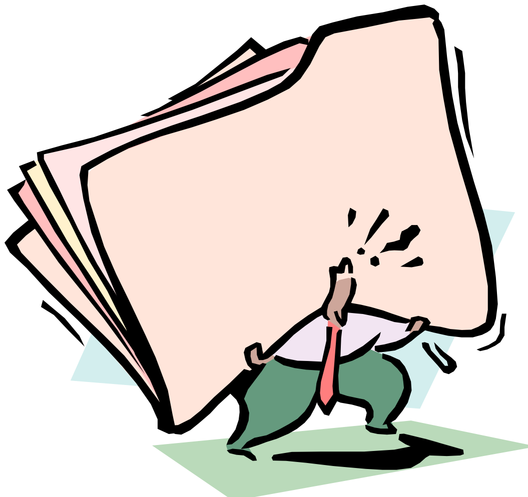
- We want to capture in an abstract way, notions of locality, of mobility, and of ability to cross barriers.
- An ambient is a place, delimited by a boundary, where computation happens.
- Ambients have a name, a collection of local processes, and a collection of subambients.
- Ambients can move in an out of other ambients, subject to capabilities that are associated with ambient names.
- Ambient names are unforgeable (as in π and spi).

The Ambient Calculus

P ::=	Processes
$(\nu n)\ P$	new name n in a scope
$\mathbf{0}$	inactivity
$P \mid P'$	parallel
$!P$	replication
$M[P]$	ambient
$M.P$	exercise a capability
$(n).P$	input locally, bind to n
$\langle M \rangle$	output locally (async)

M ::=	Messages
n	name
in M	entry capability
out M	exit capability
open M	open capability
ε	empty path
$M.M'$	composite path

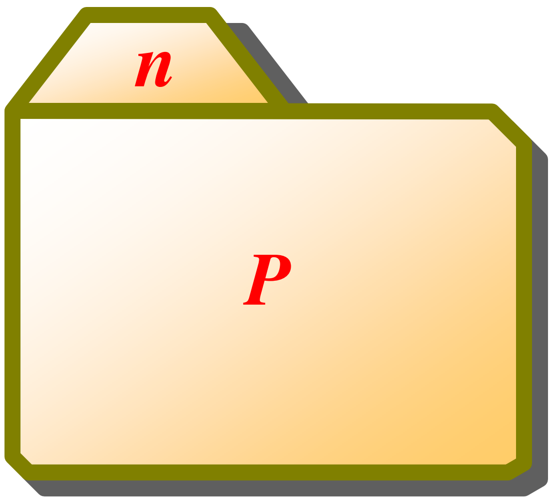
The Folder Calculus



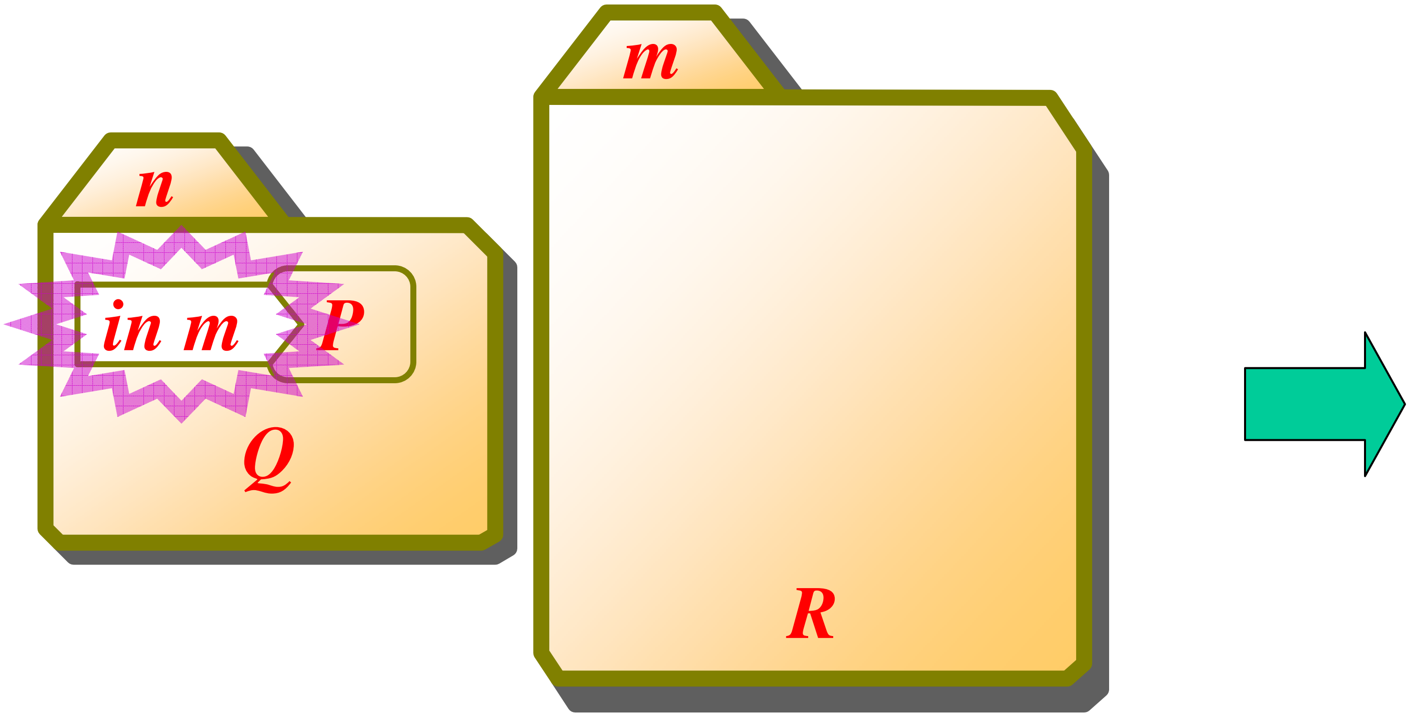
- A graphical office metaphor to explain the ambient calculus.
- A precise metaphor, isomorphic to the formal ambient calculus.
- Based on wide-area computation principles: locality, mobility, nested domains, asynchronous communication, authentication.

Folders (Nested Domains)

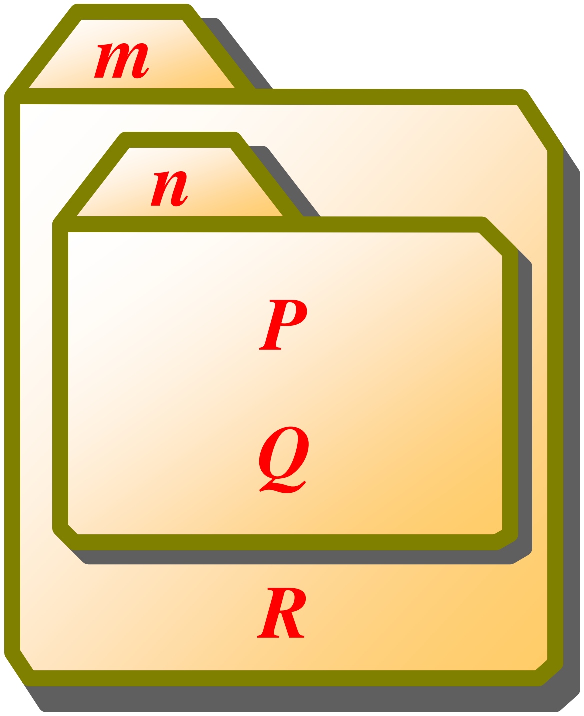
- Folders have a folder name *n*,
- And have active contents *P*, including:
  - Hierarchical data, and computations (processes/“gremlins”).
  - Primitives for mobility and communication.



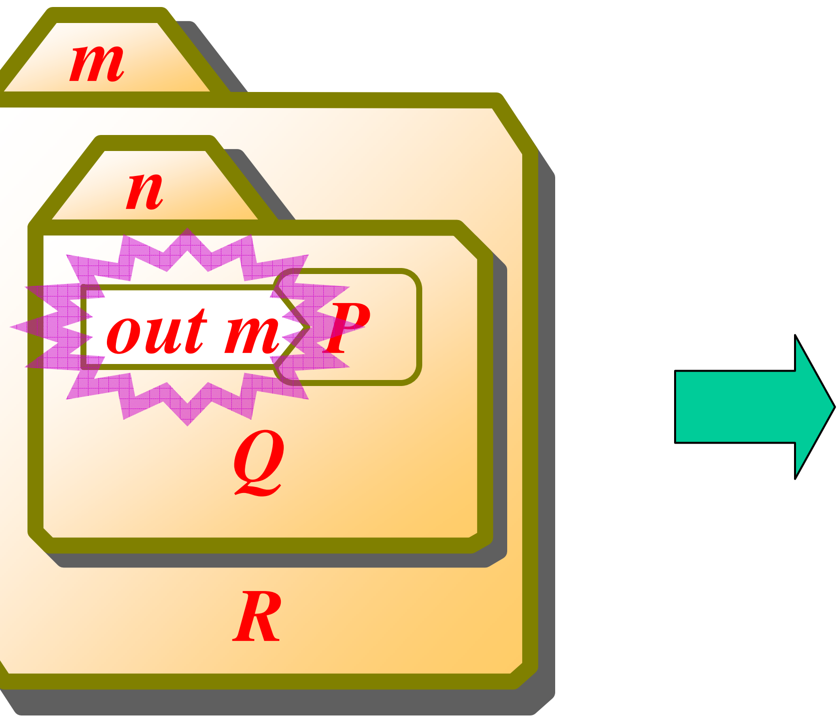
Enter Reduction (Mobility)



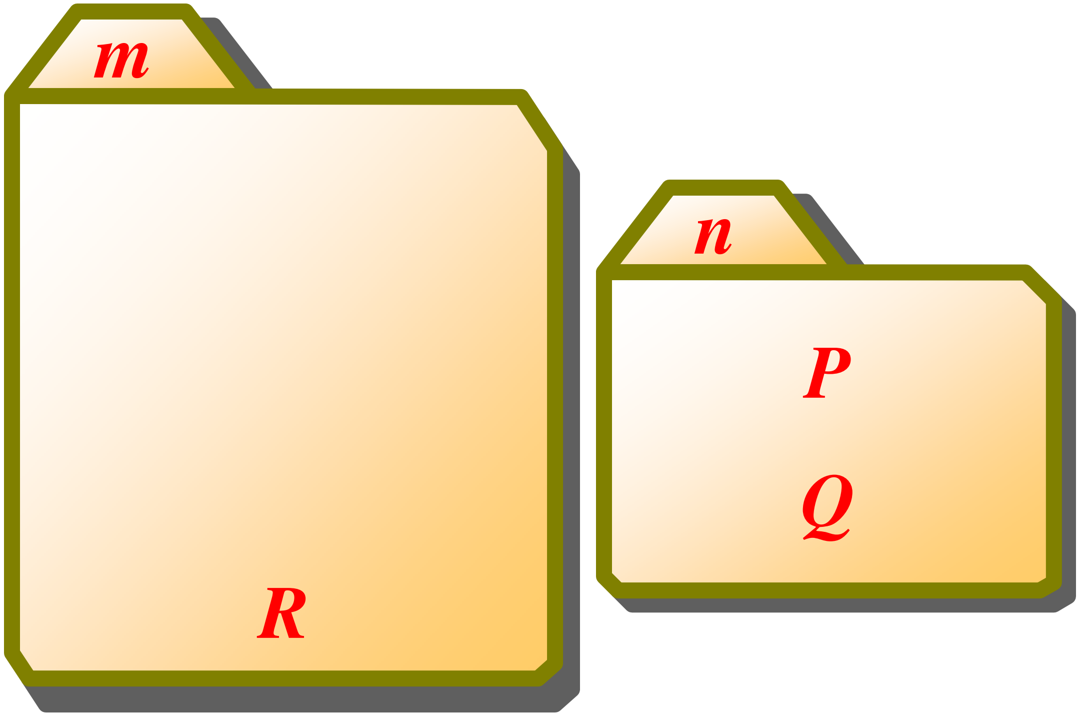
Enter Reduction (Mobility)



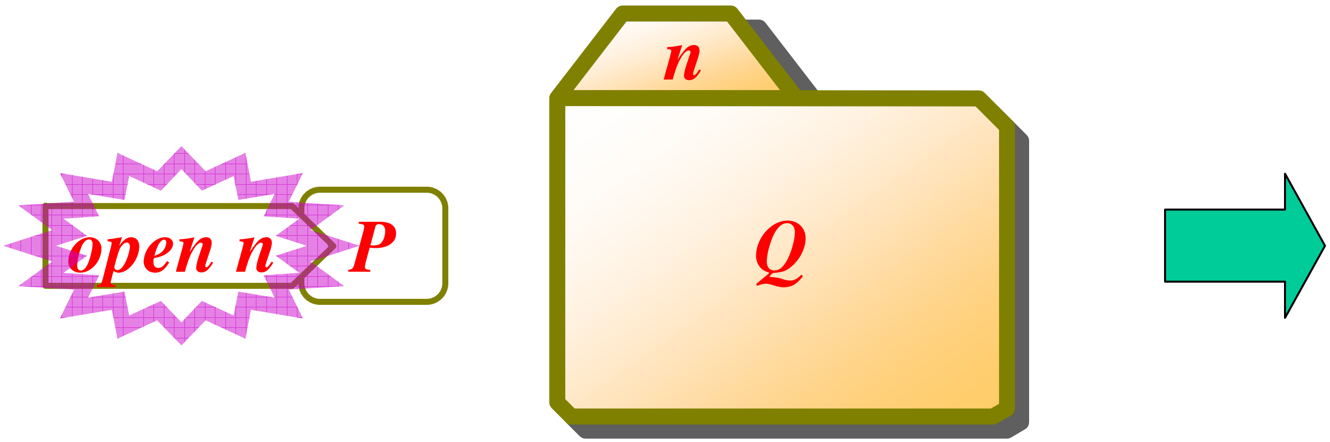
Exit Reduction (Mobility)



Exit Reduction (Mobility)



Open Reduction (Assimilation)

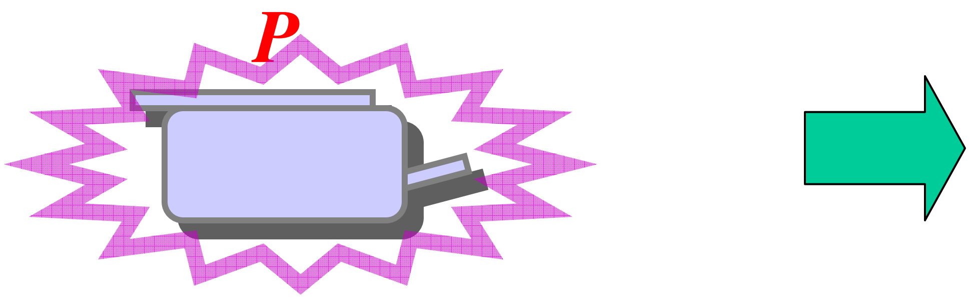


Open Reduction (Assimilation)

P ***Q***

Copy Reduction (Iteration/Recursion)

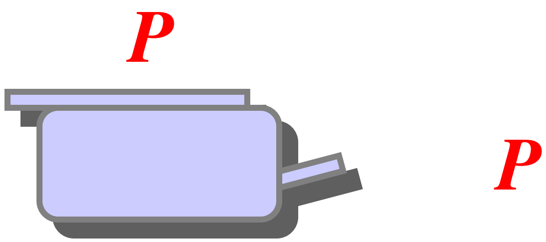
- Unlimited (on-demand) replication:



- *P* can be any folder or configuration, but it is not “running” until it is replicated.

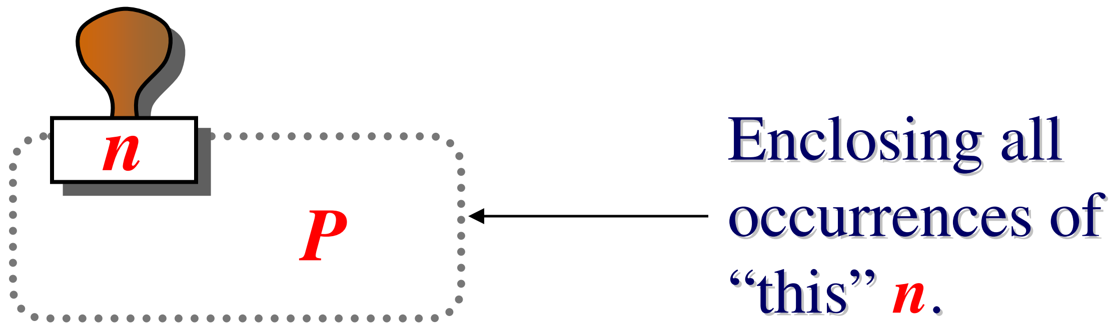
Copy Reduction (Iteration/Recursion)

- Unlimited (on-demand) replication:

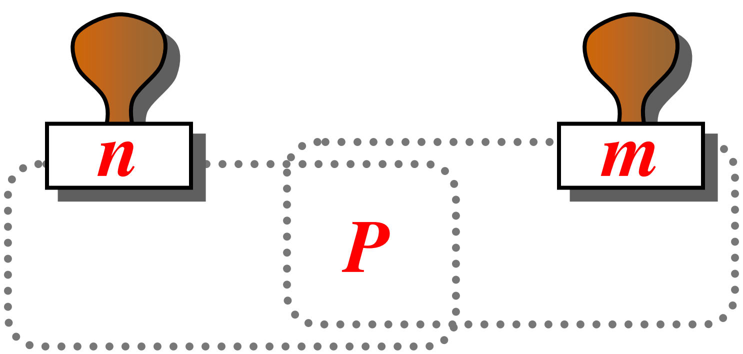


- *P* can be any folder or configuration, but it is not "running" until it is replicated.

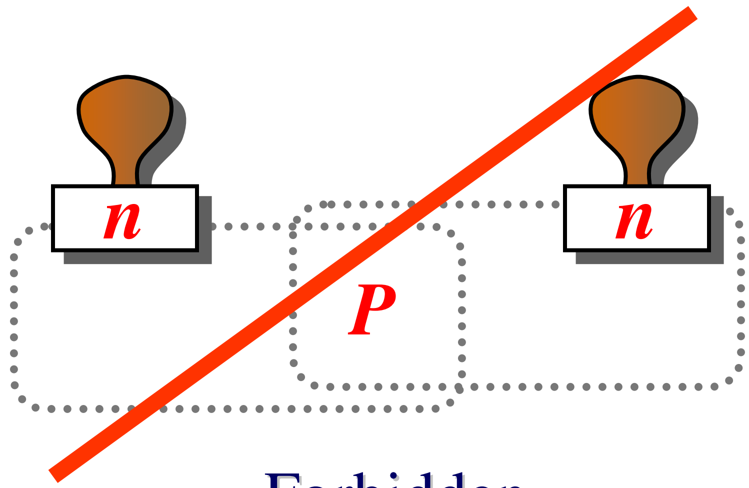
Rubber Stamps (Authentication)



- Rubber stamps give authenticity to folders.
  - (Copiers are unable to accurately replicate rubber stamps.)
- Scoping Rules:



Allowed



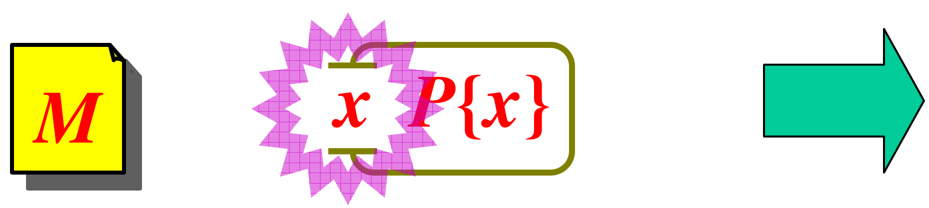
Forbidden

Post-It Notes (Local Communication)

- A Post-It Note (Nameless file / Asynchronous message):

- A gremlin grabbing (reading and removing) a note:

- Read reduction:



Post-It Notes (Local Communication)

- A Post-It Note (Nameless file / Asynchronous message):

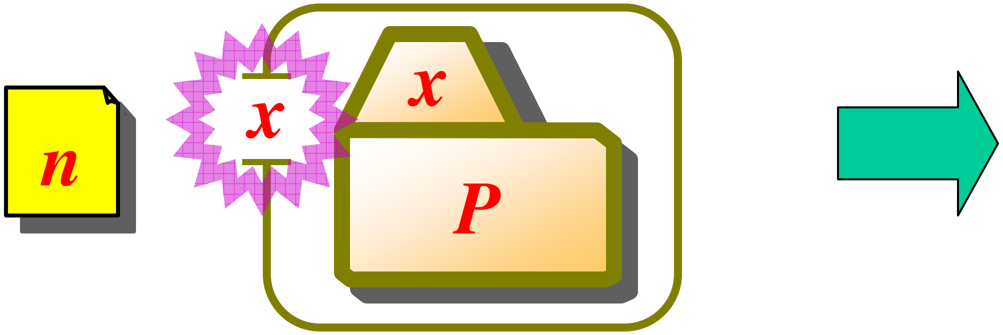
- A gremlin grabbing (reading and removing) a note:

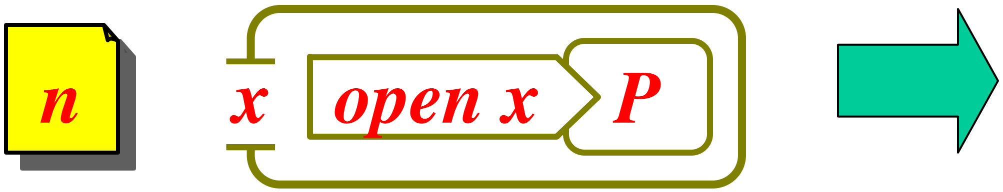
- Read reduction:

$$P\{M\}$$

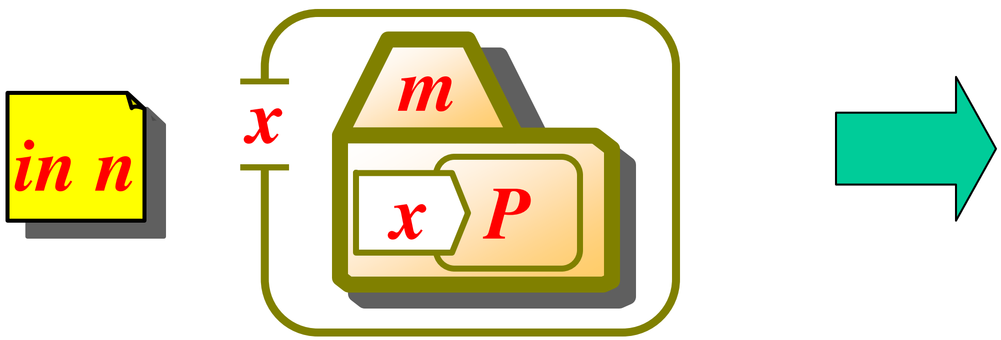
Messages (Names or Capabilities)

- A message M can be either:
  - The name of a folder (danger: spoofing, killing):



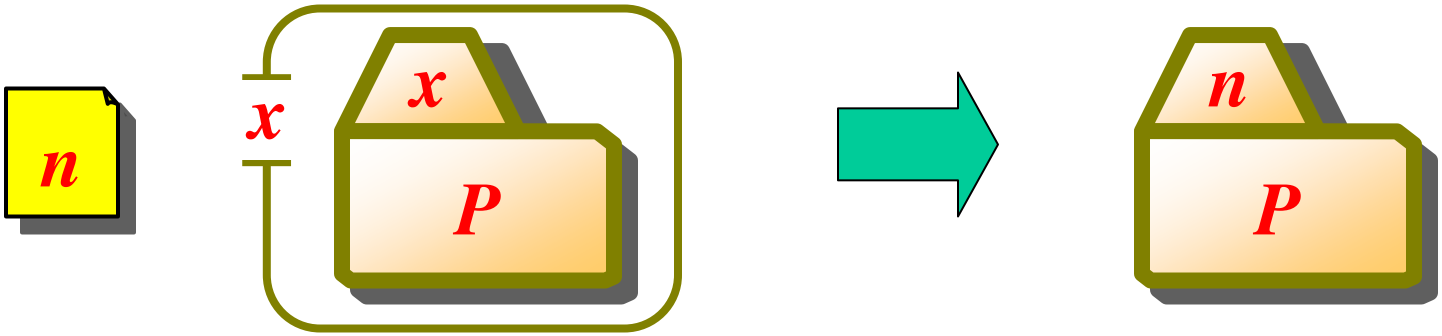


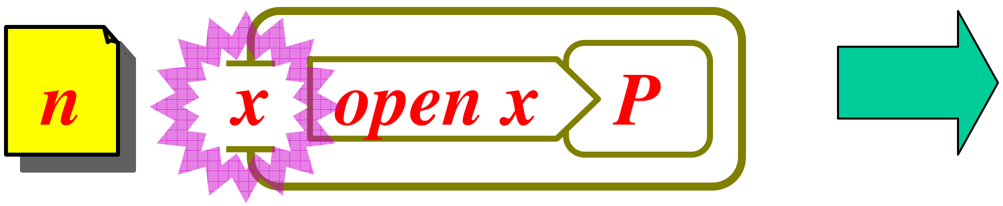
  - A capability (no danger of recovering the full name):



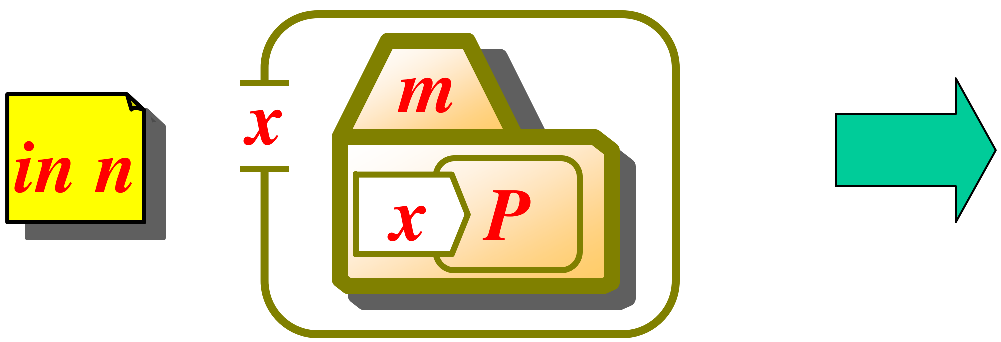
Messages (Names or Capabilities)

- A message M can be either:
  - The name of a folder (danger: spoofing, killing):



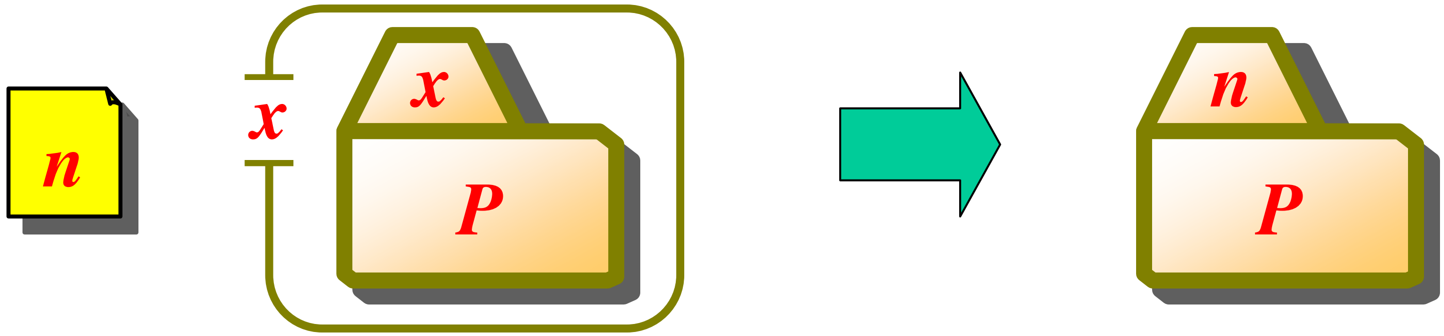


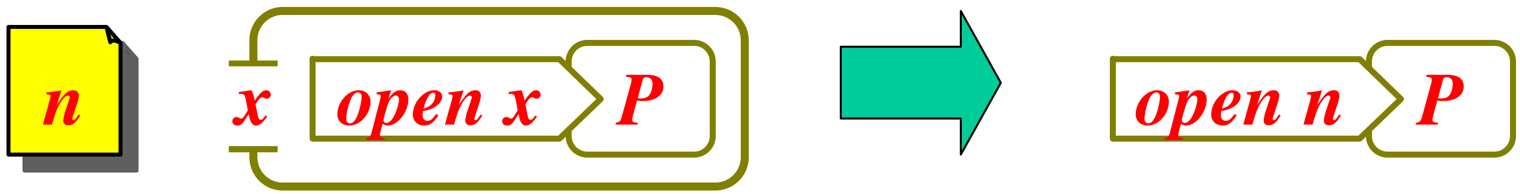
  - A capability (no danger of recovering the full name):



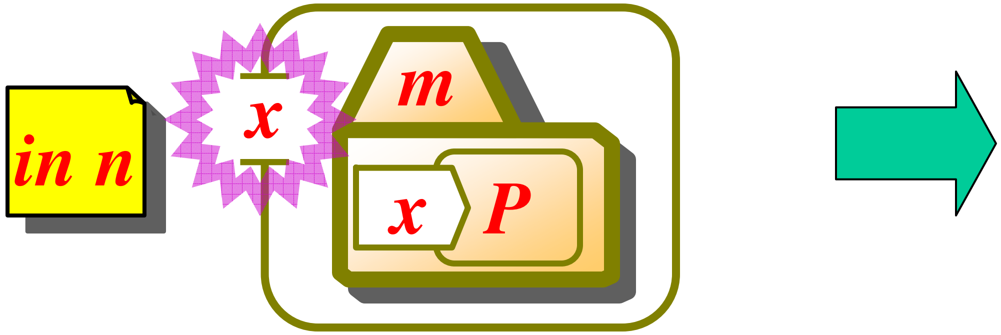
Messages (Names or Capabilities)

- A message M can be either:
  - The name of a folder (danger: spoofing, killing):



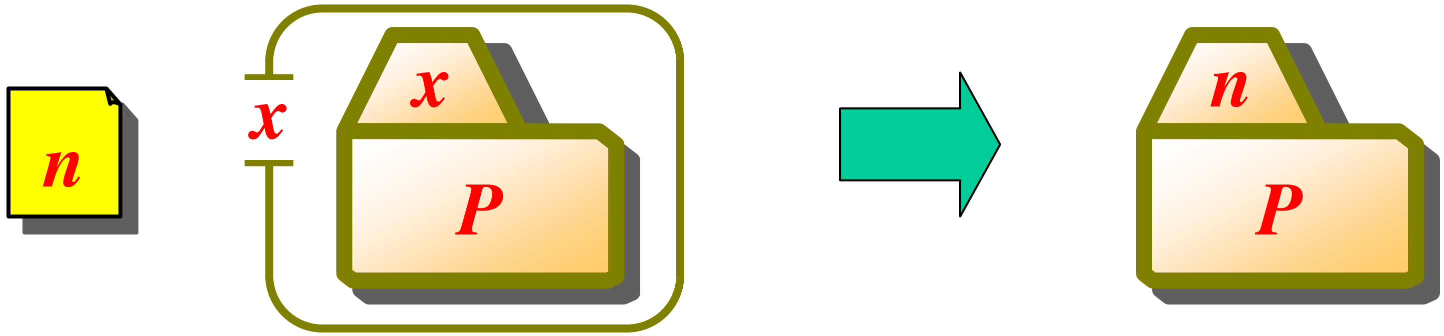


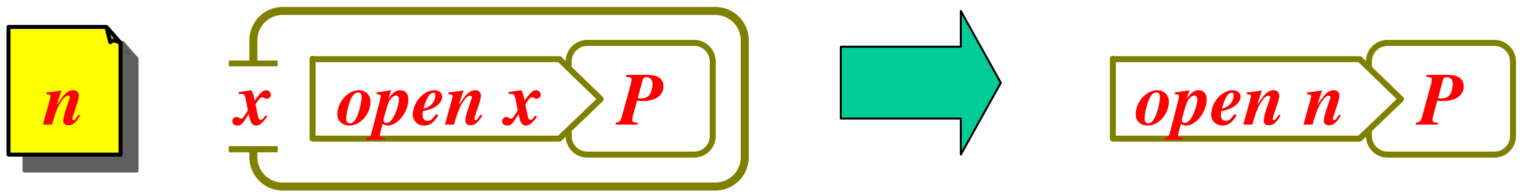
  - A capability (no danger of recovering the full name):



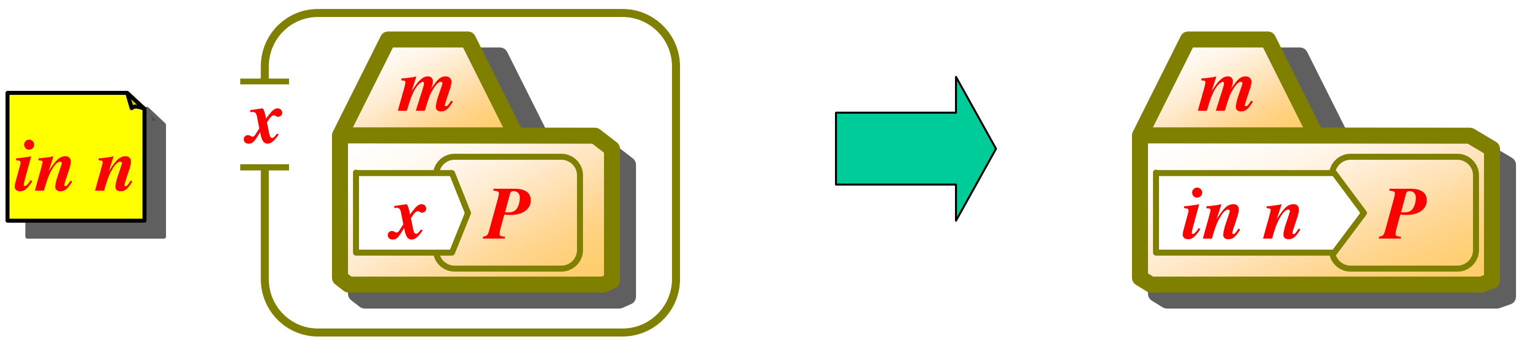
Messages (Names or Capabilities)

- A message M can be either:
  - The name of a folder (danger: spoofing, killing):



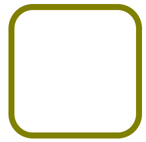


  - A capability (no danger of recovering the full name):

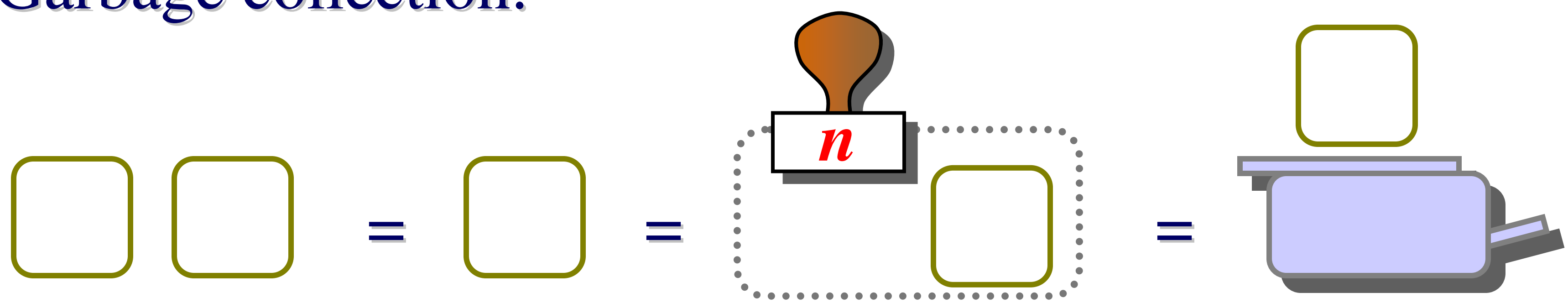


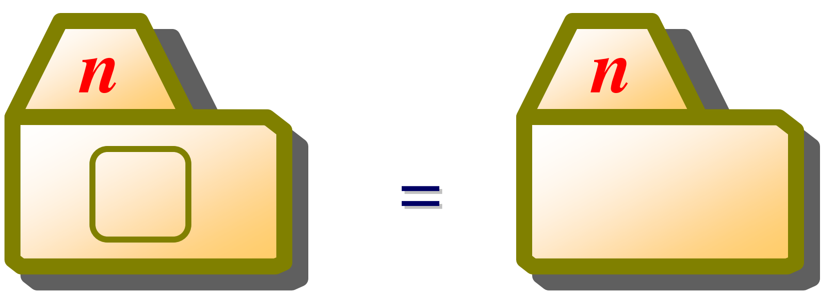
Leaves of the Syntax

- Inactive gremlin:

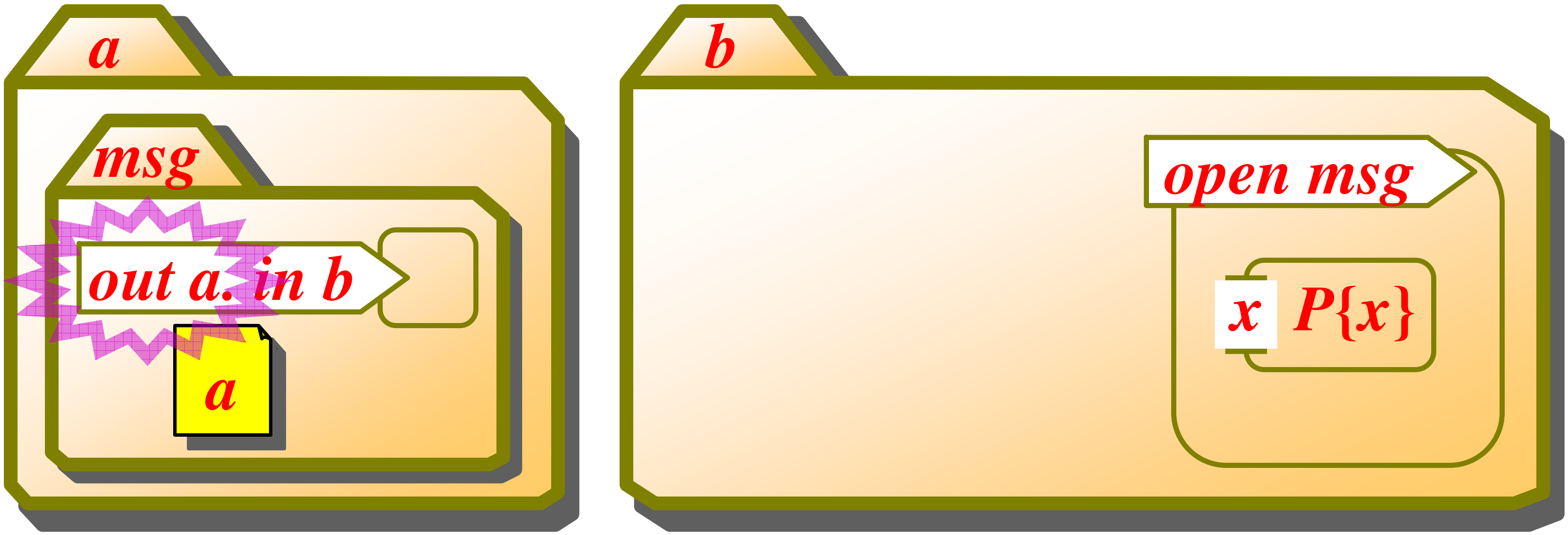


- Garbage collection:



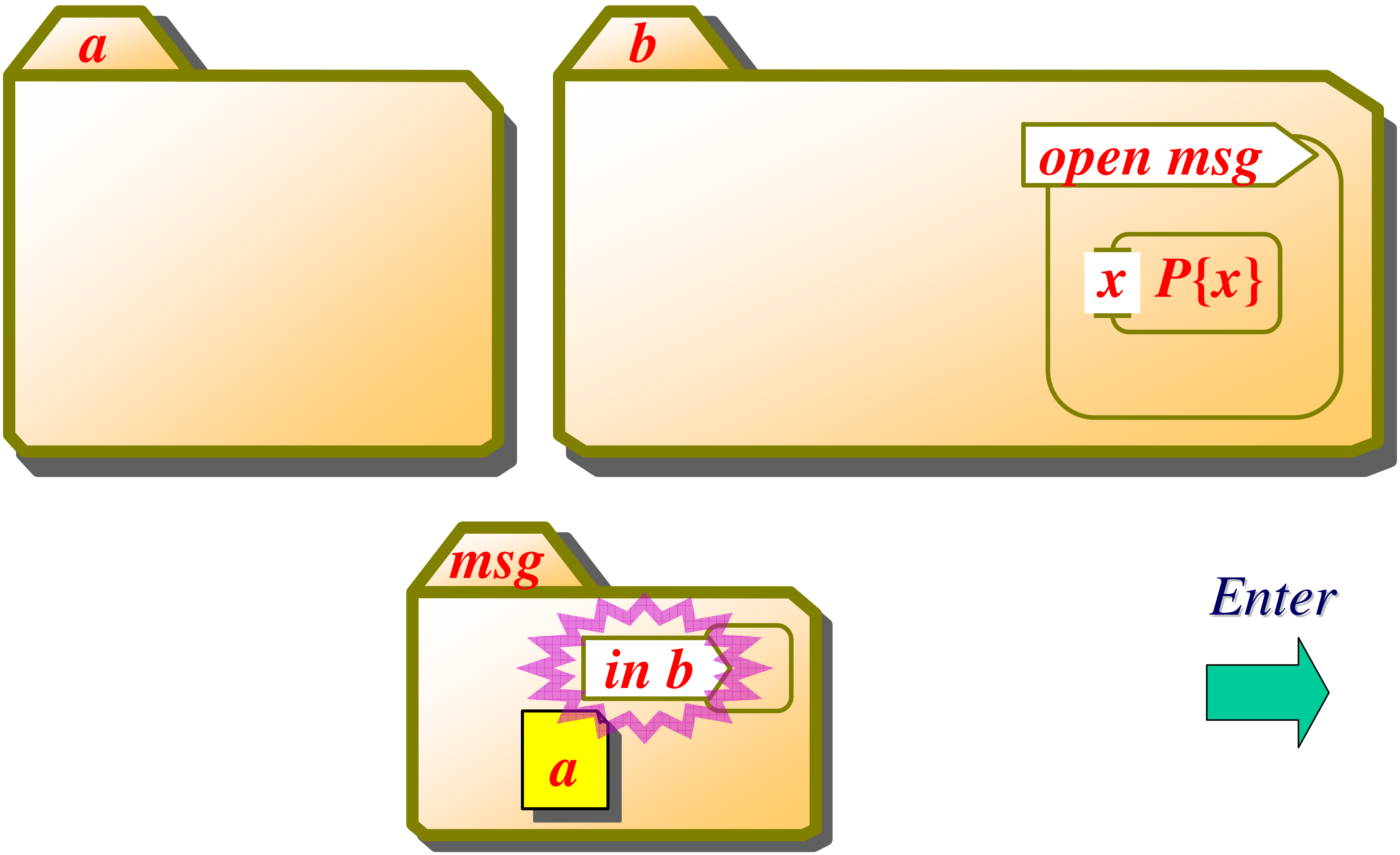


Example: Message from *a* to *b*

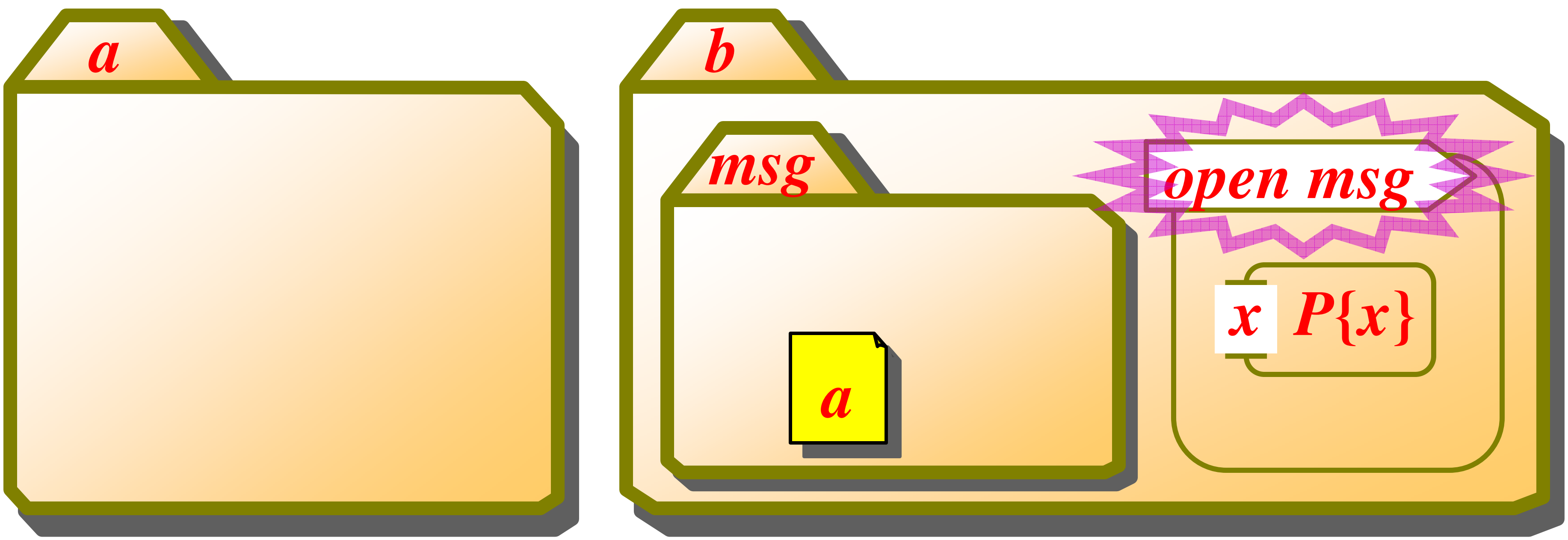


Exit

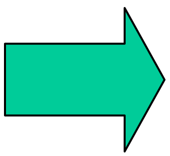
Example: Message from *a* to *b*



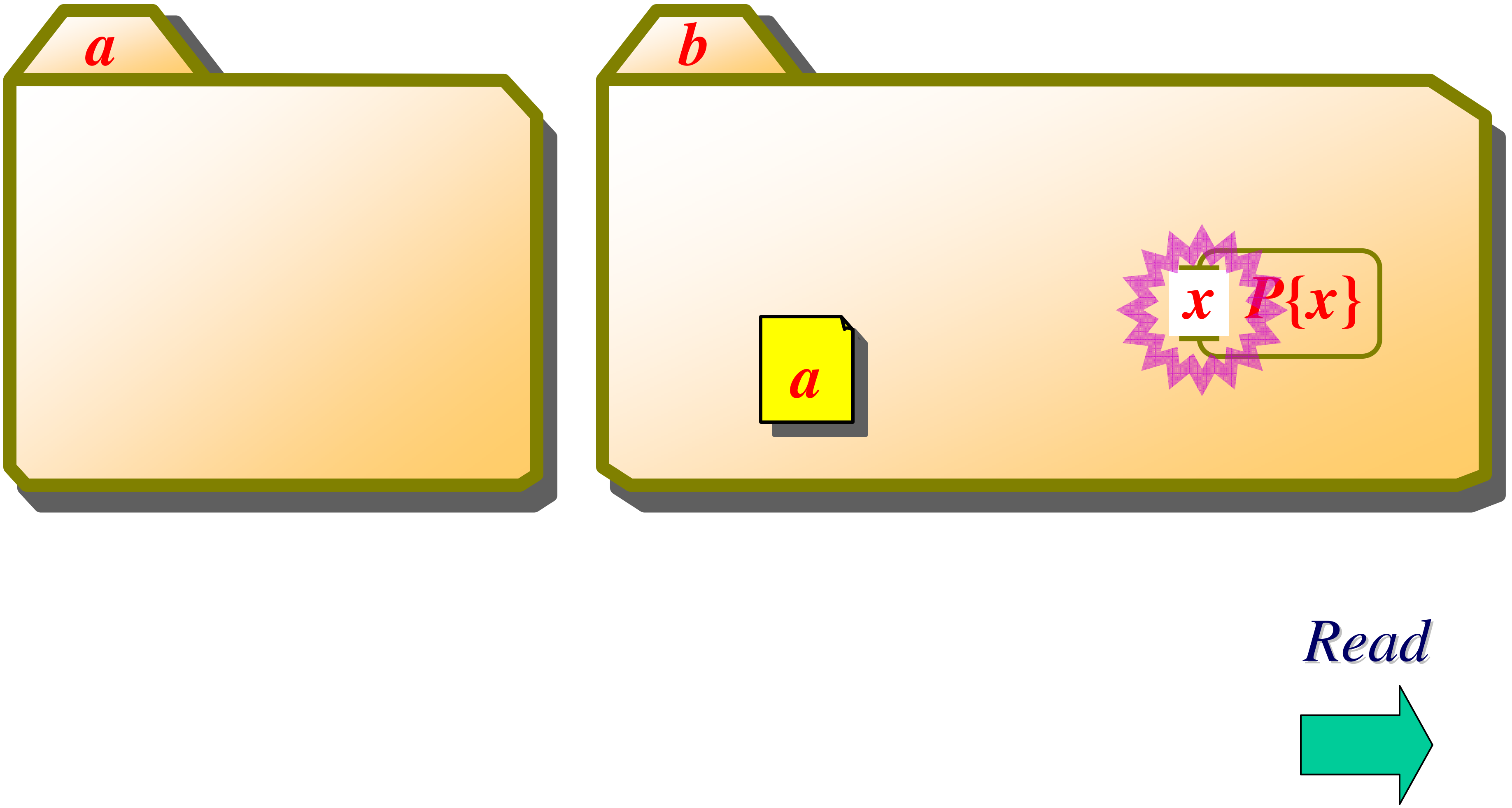
Example: Message from *a* to *b*



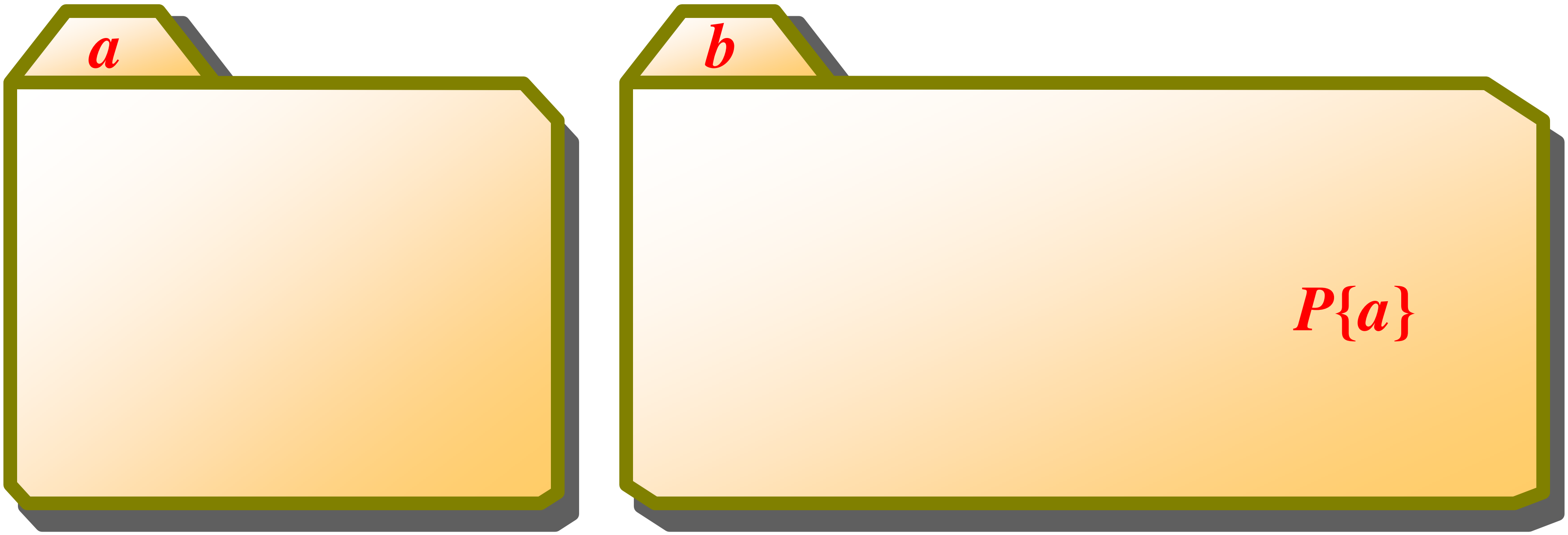
Open



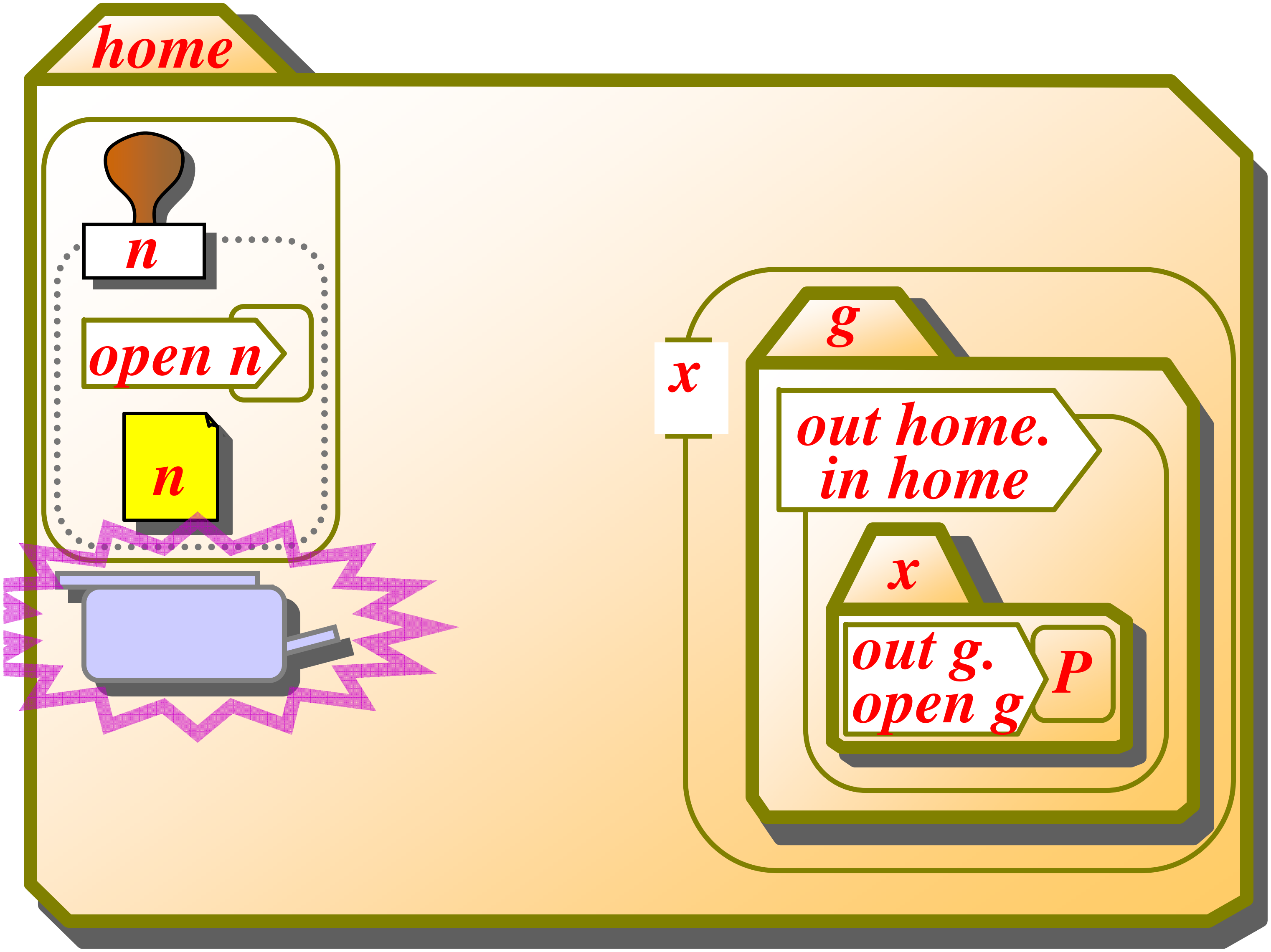
Example: Message from *a* to *b*



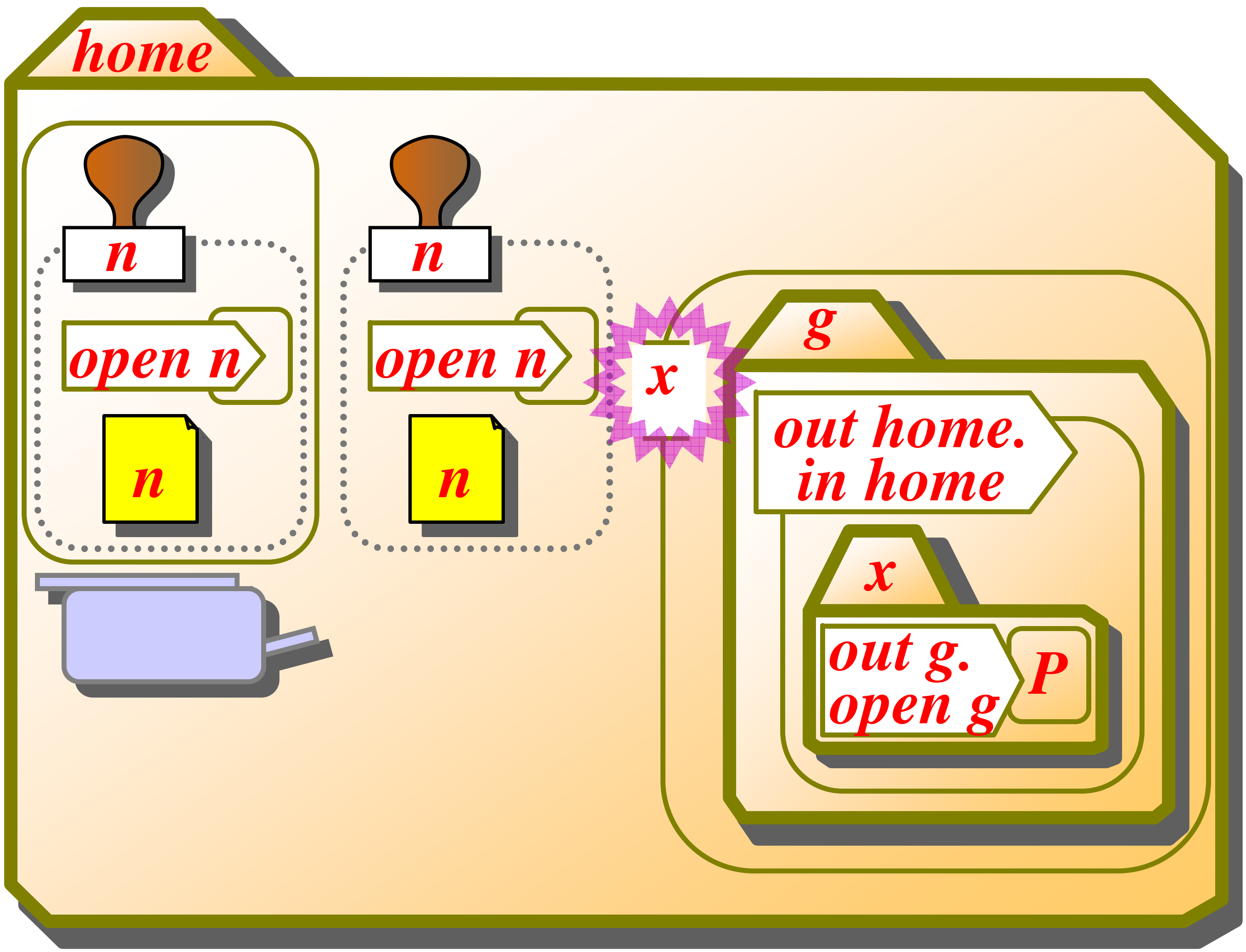
Example: Message from *a* to *b*



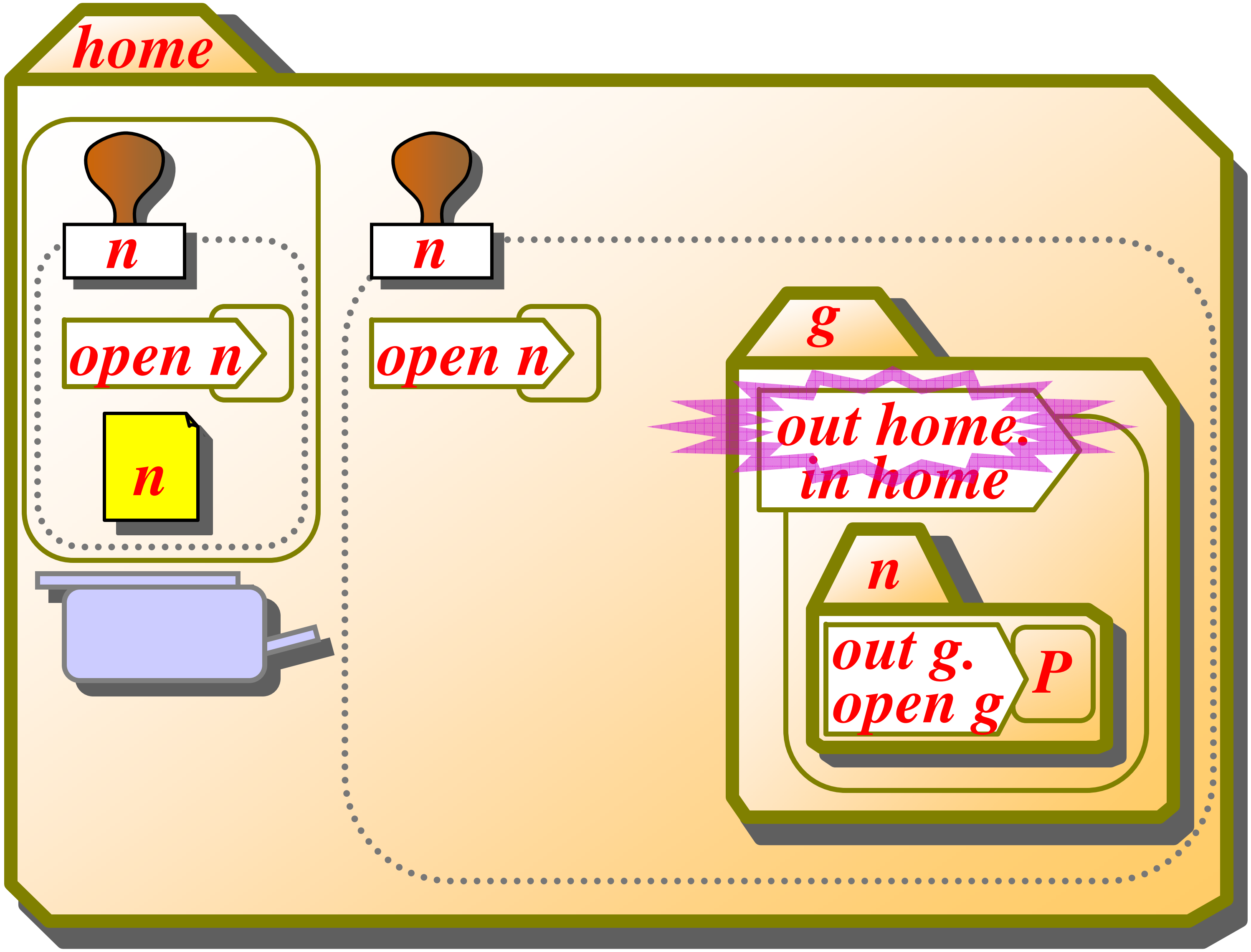
Example: Agent Authentication



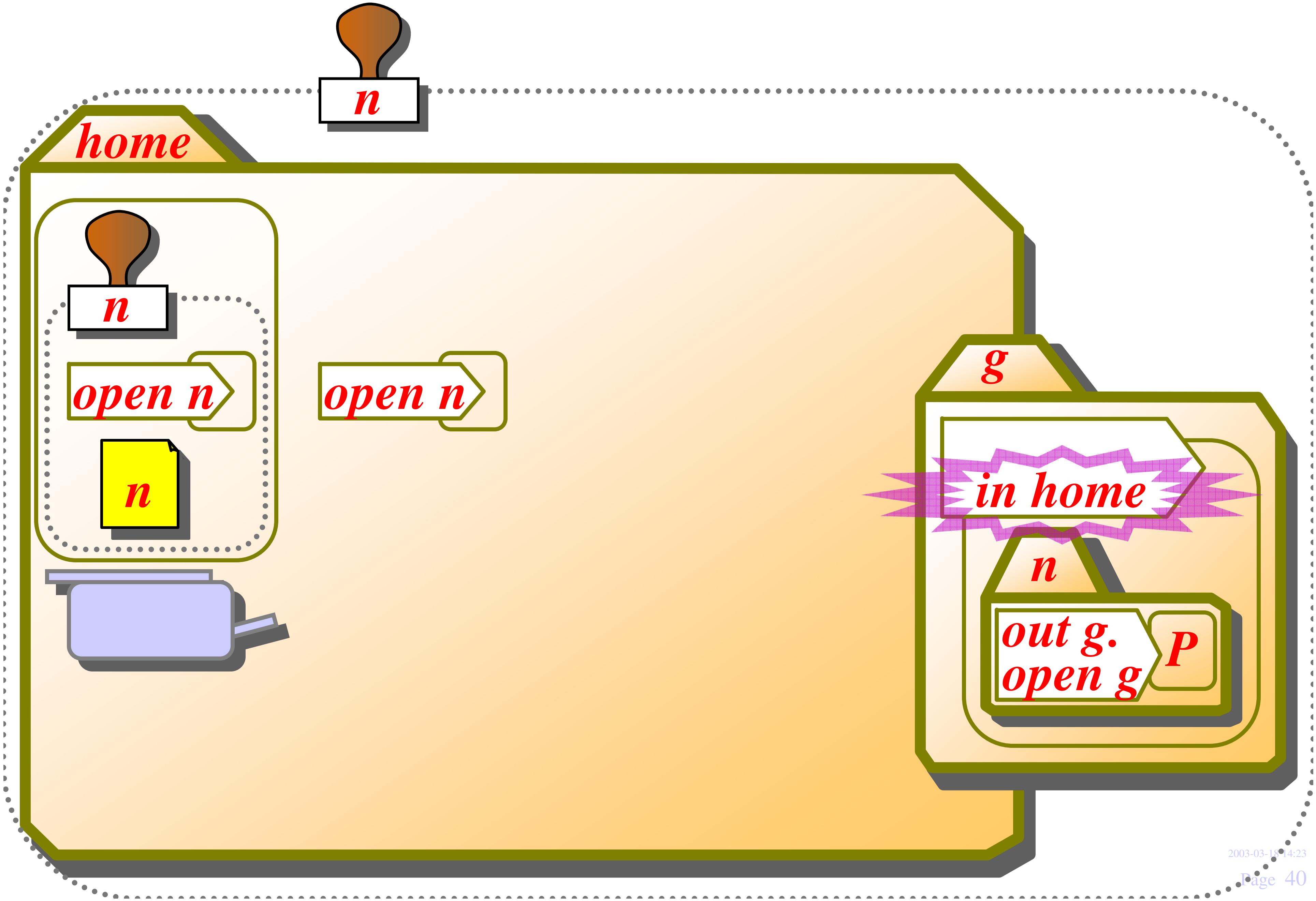
Example: Agent Authentication



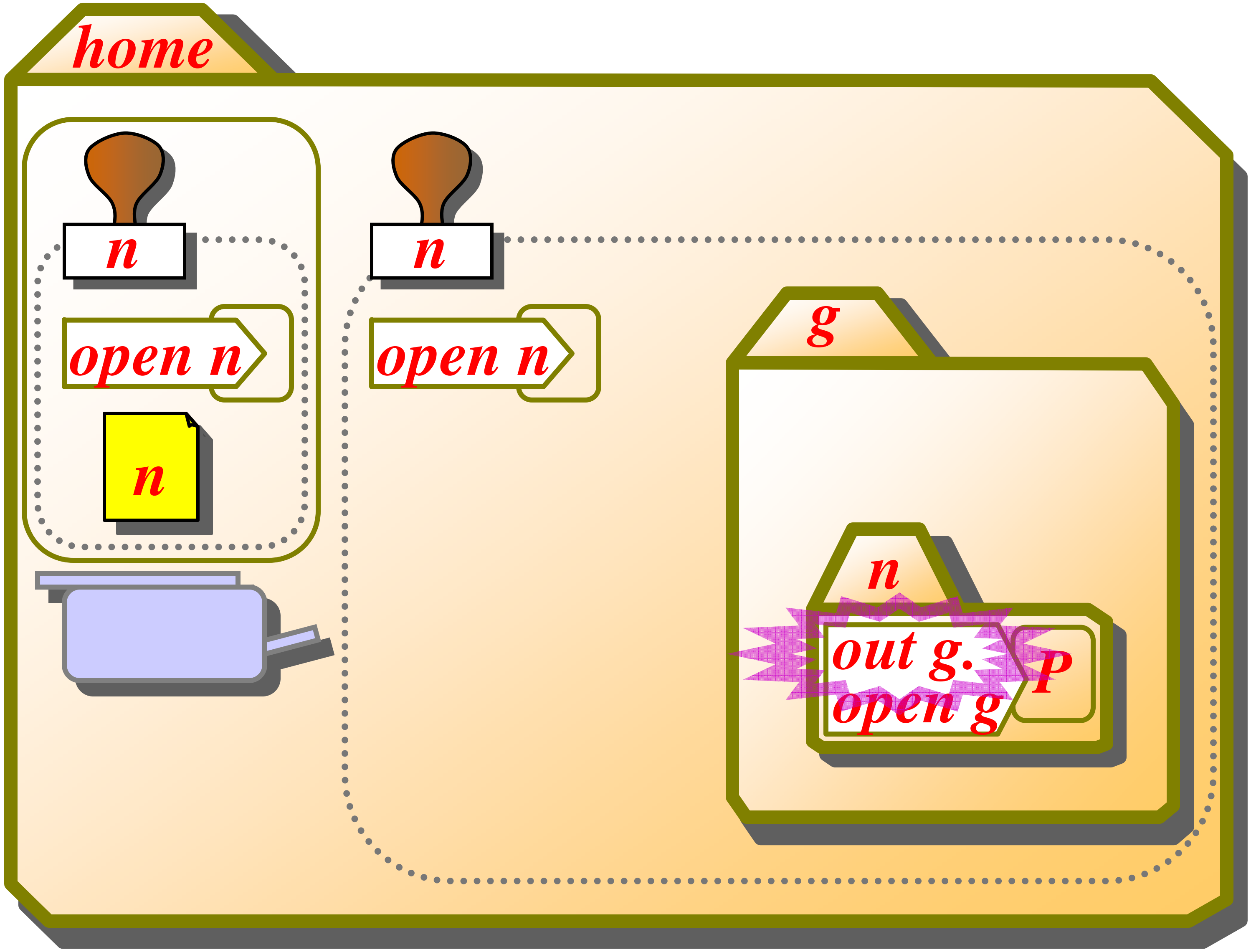
Example: Agent Authentication



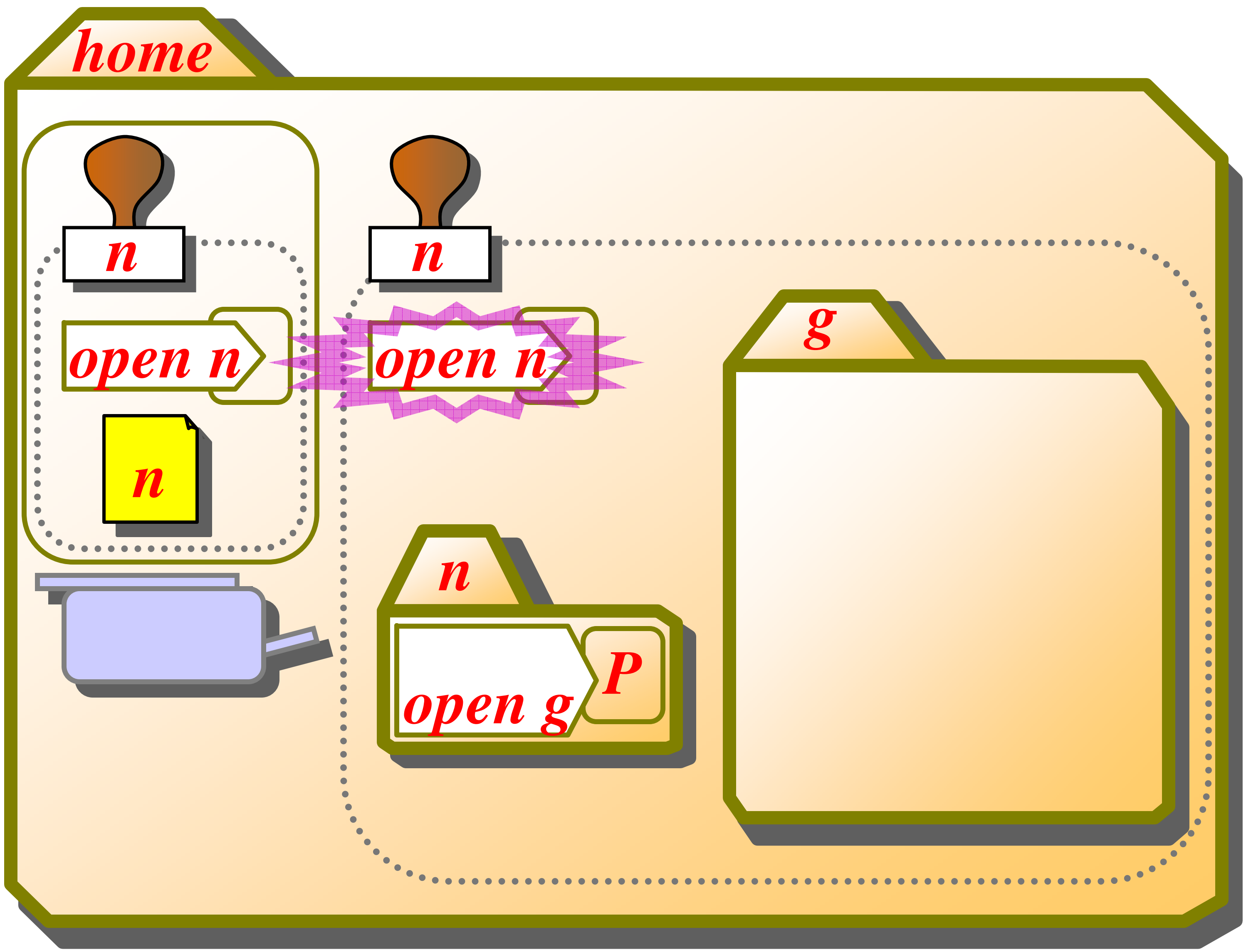
Example: Agent Authentication



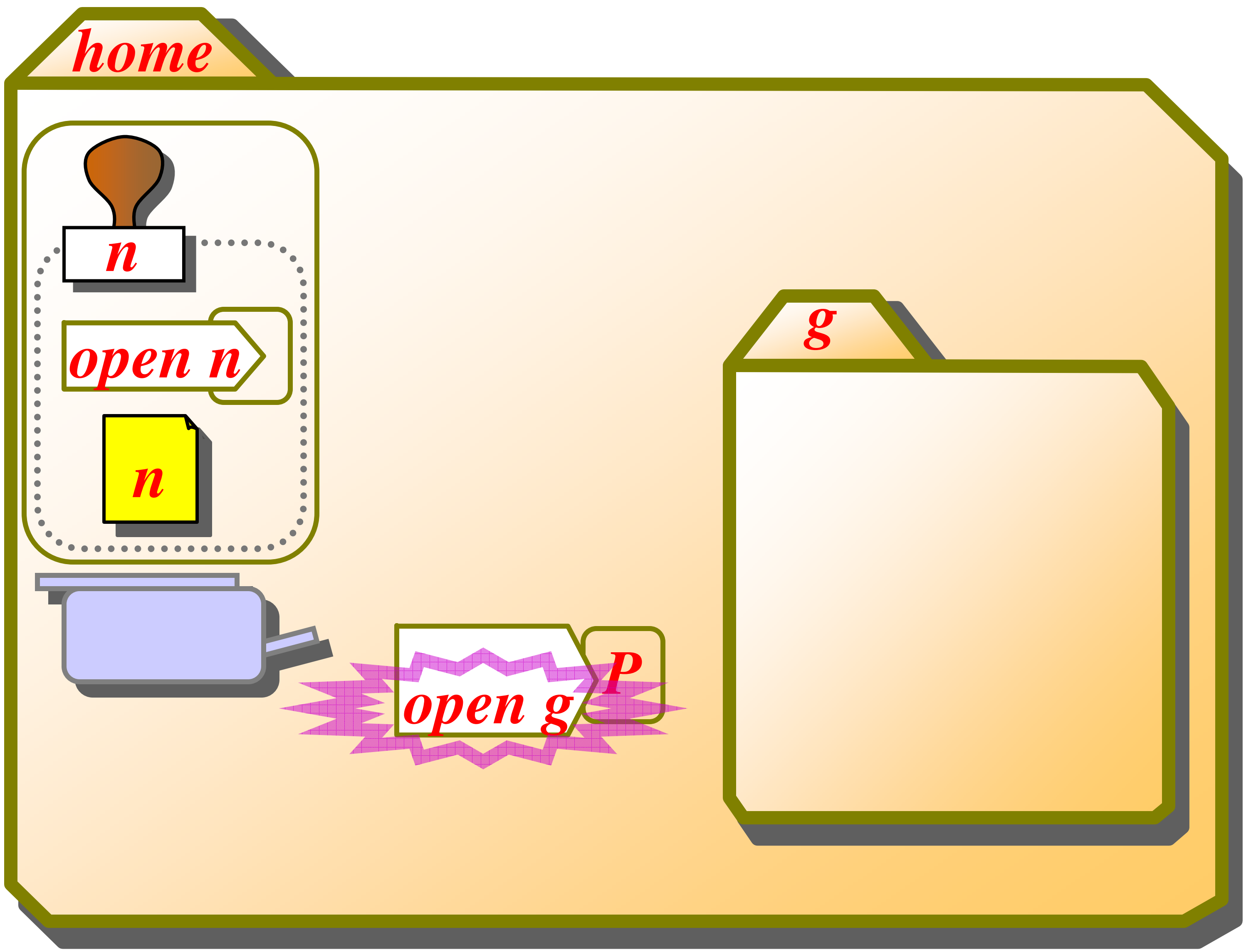
Example: Agent Authentication



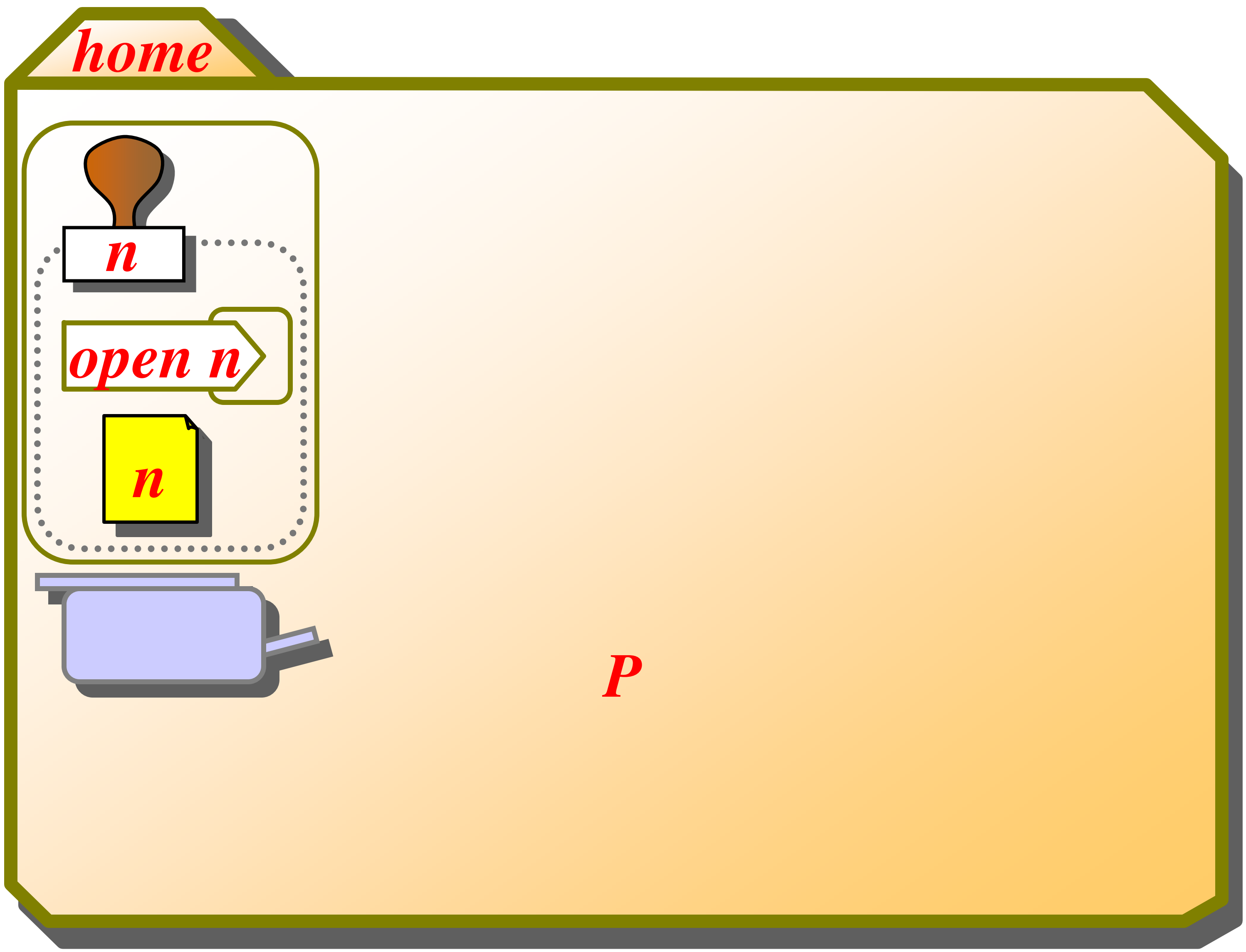
Example: Agent Authentication



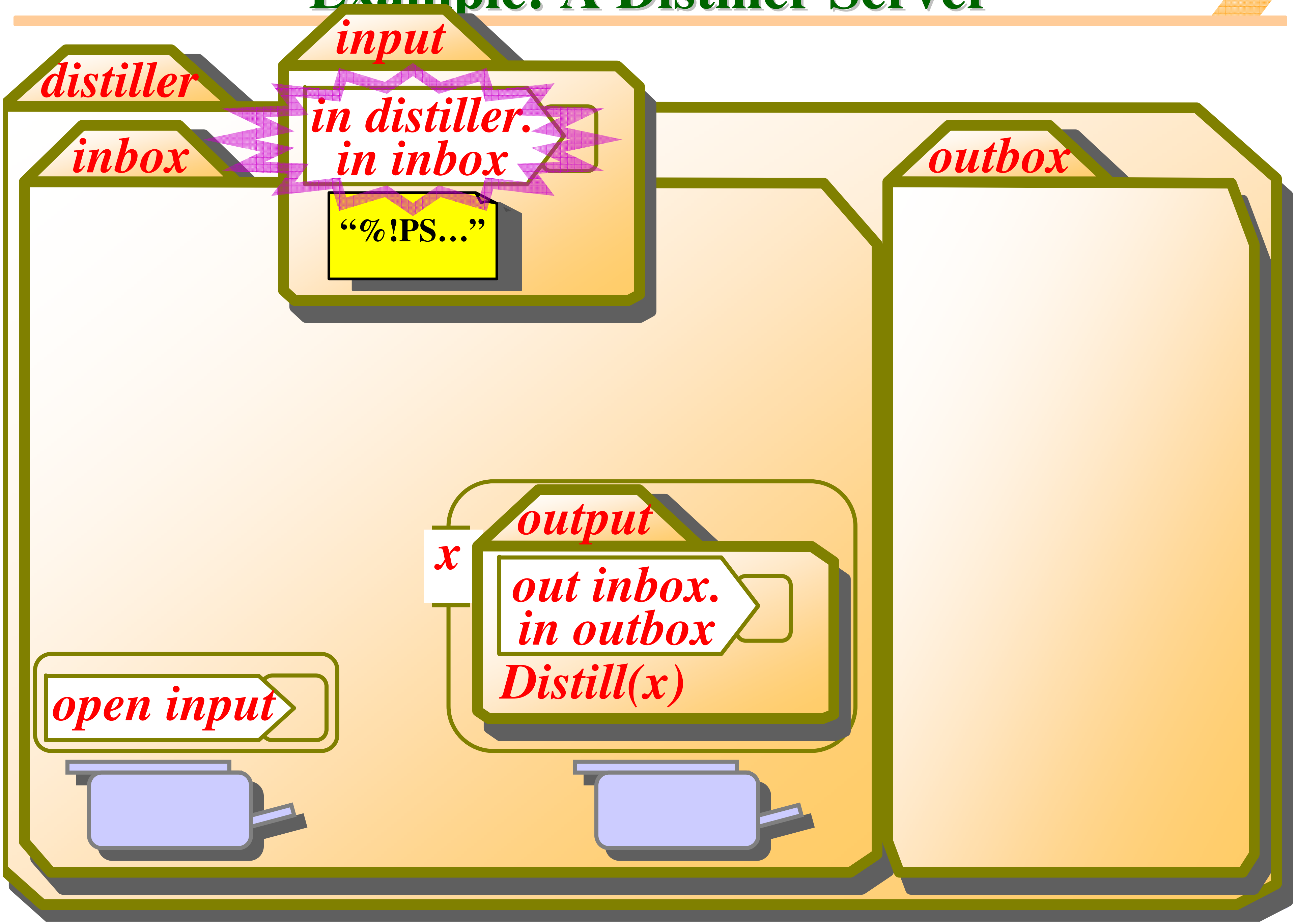
Example: Agent Authentication



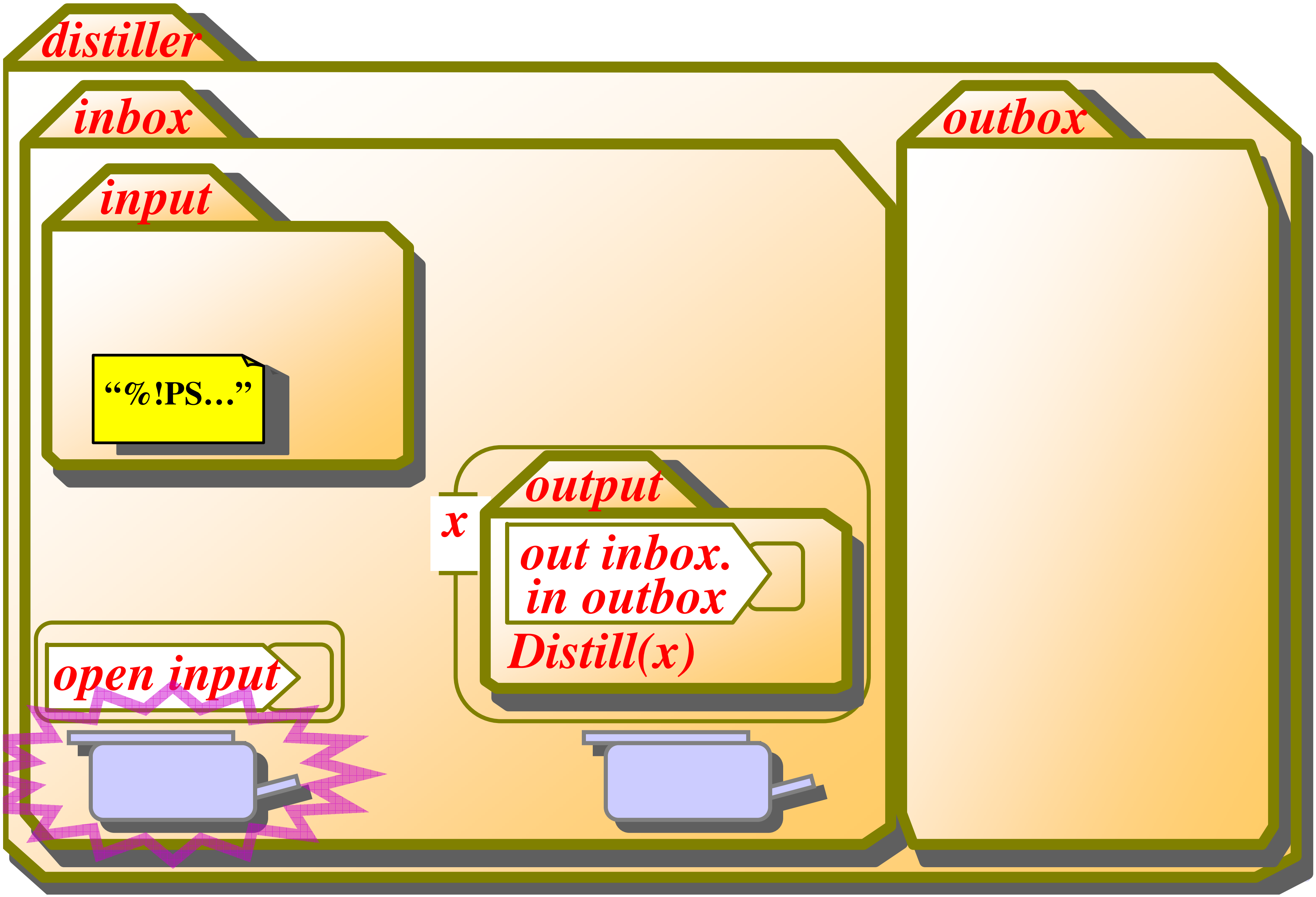
Example: Agent Authentication



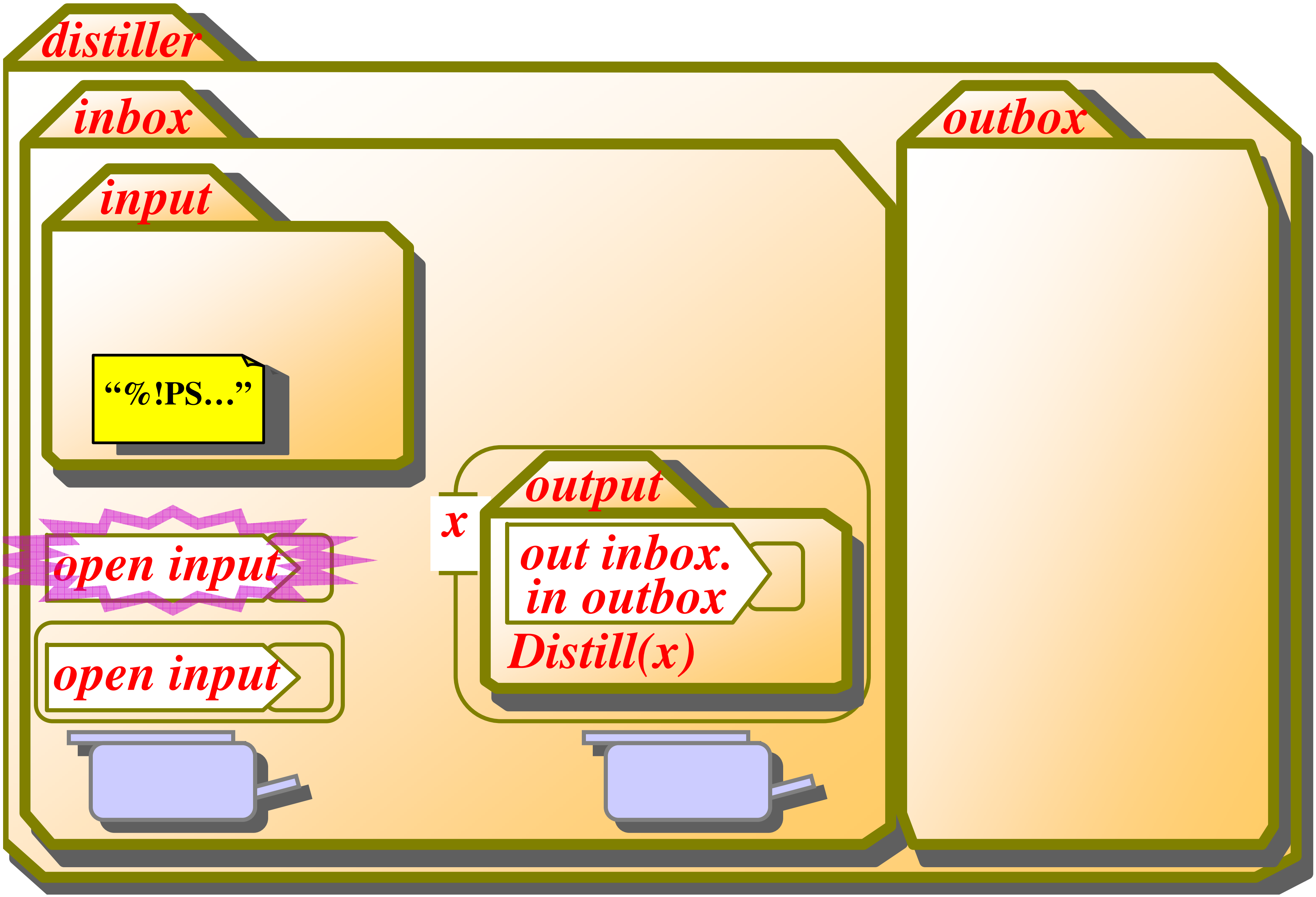
Example: A Distiller Server



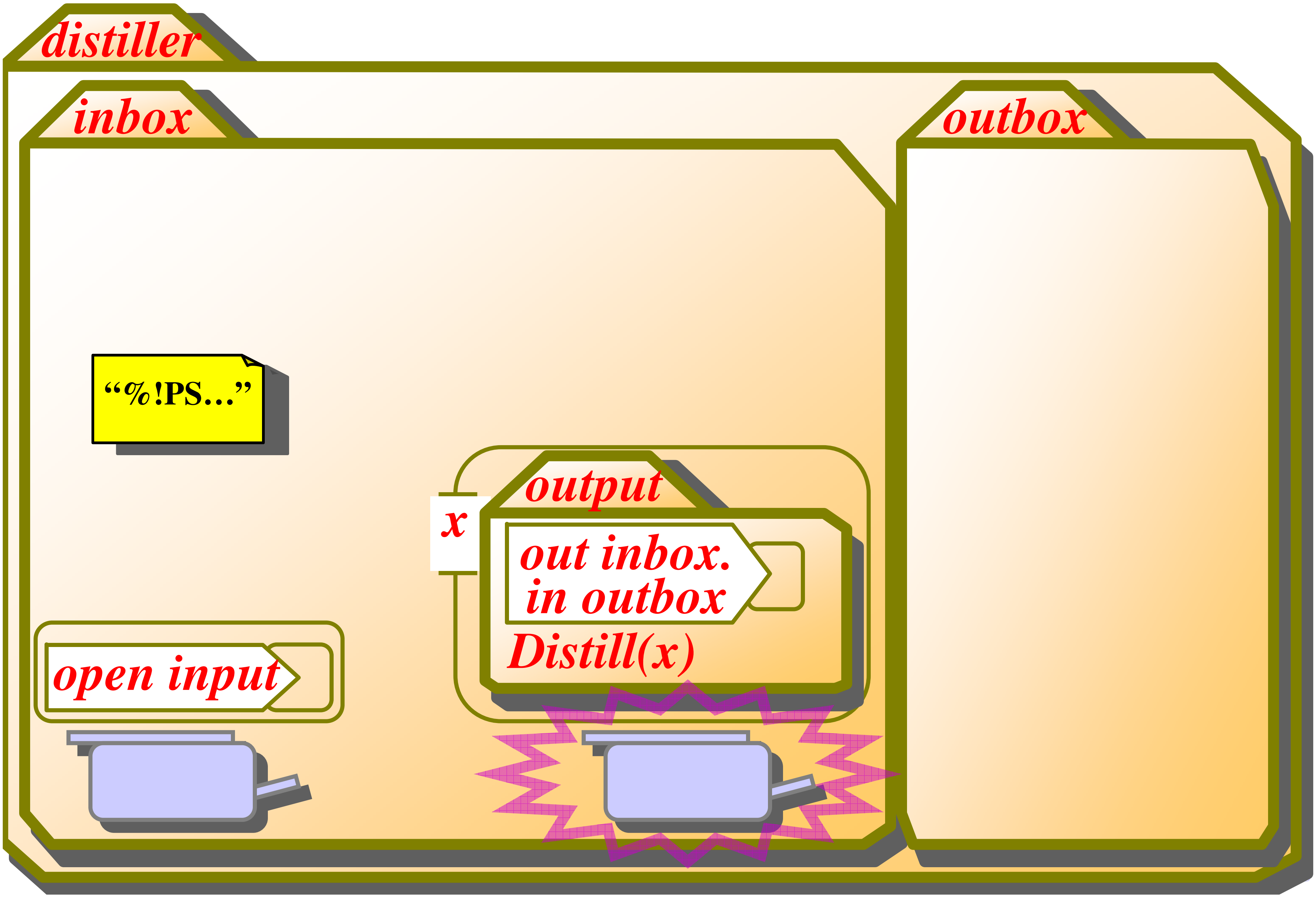
Example: A Distiller Server



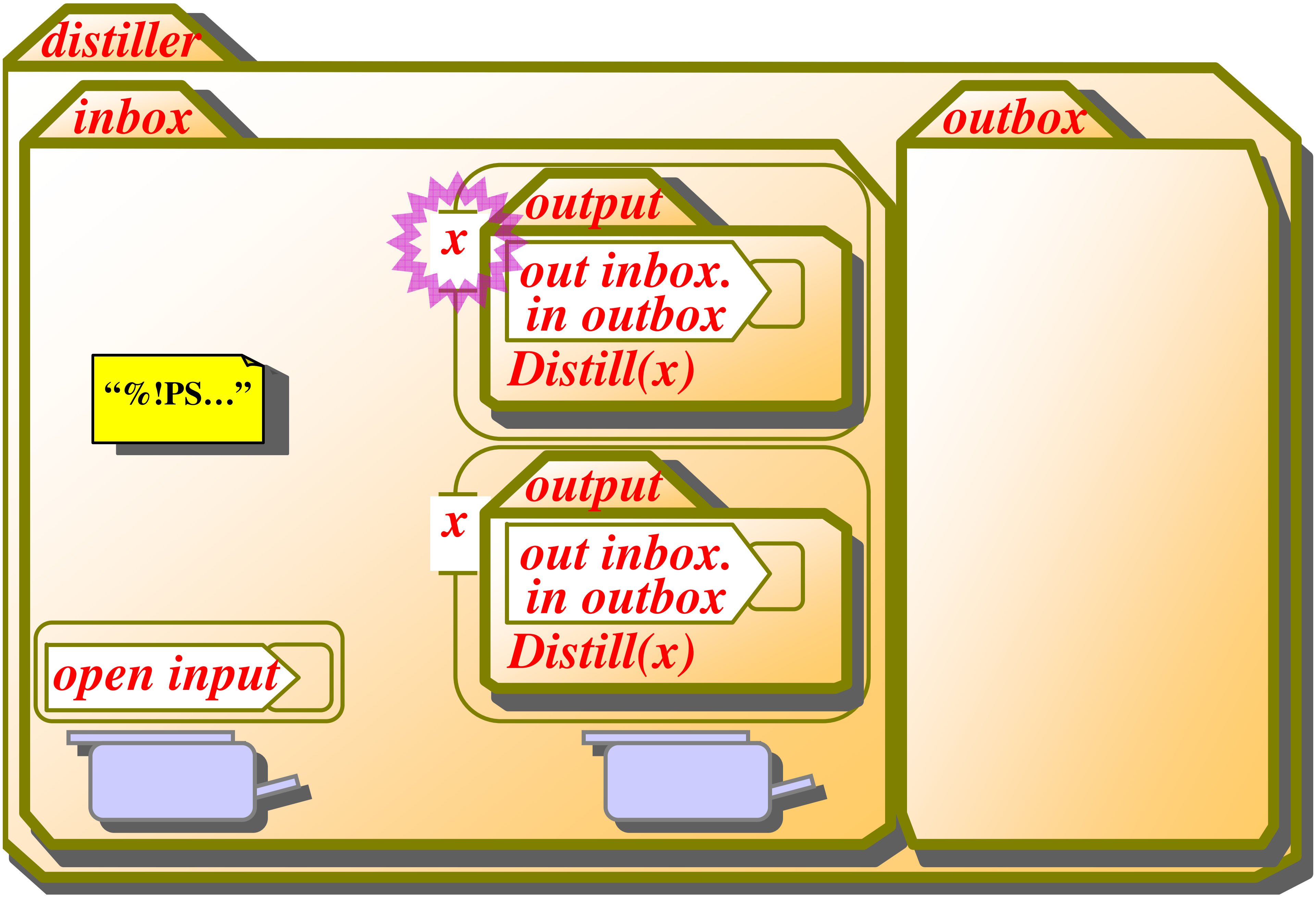
Example: A Distiller Server



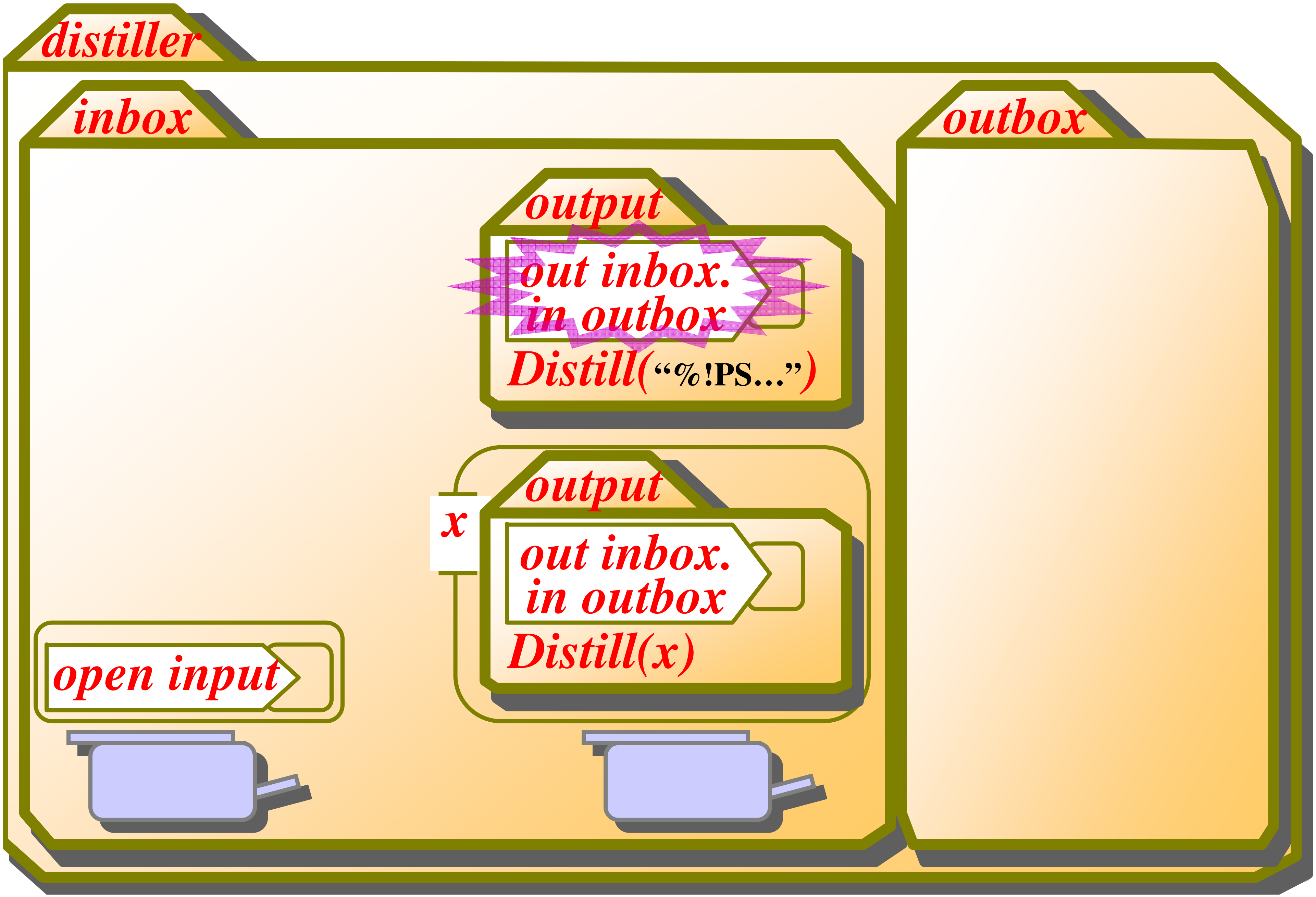
Example: A Distiller Server



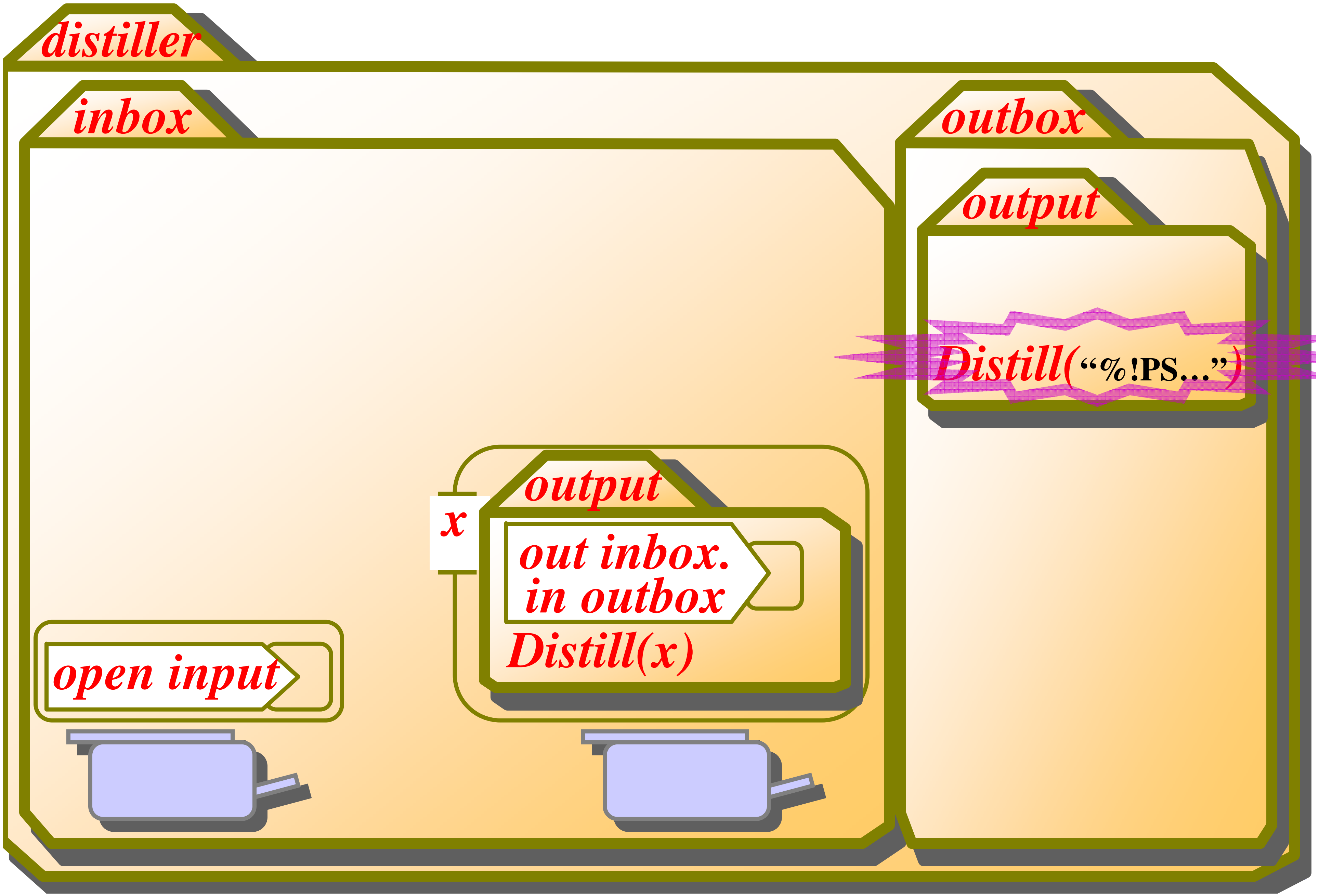
Example: A Distiller Server



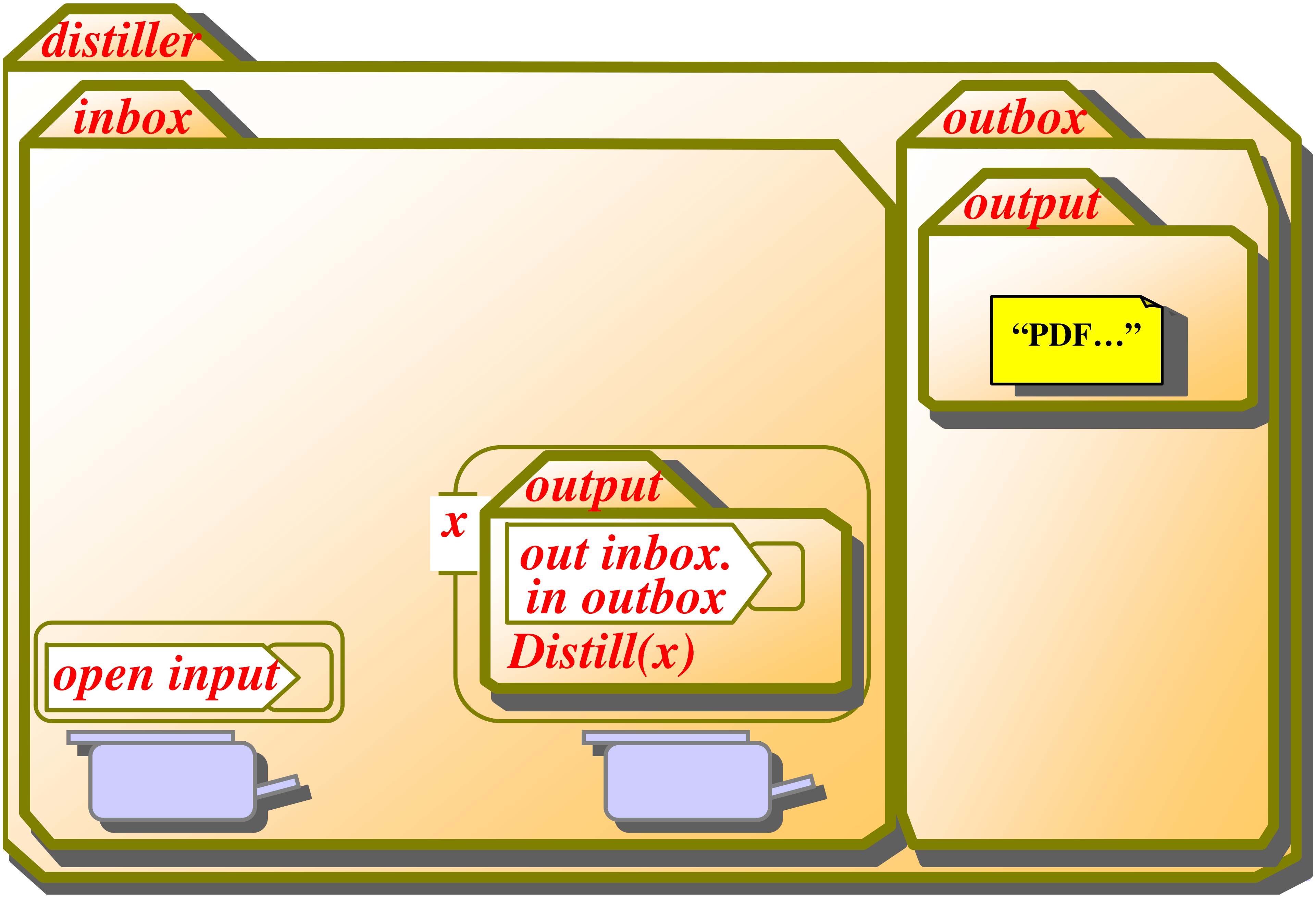
Example: A Distiller Server



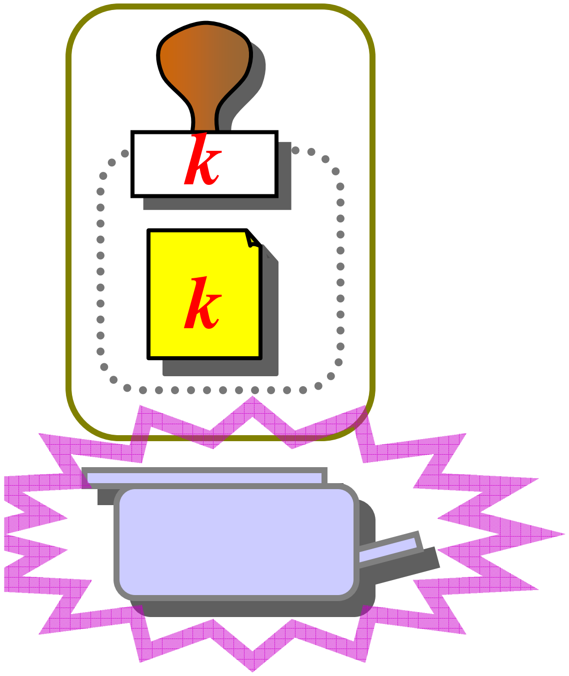
Example: A Distiller Server

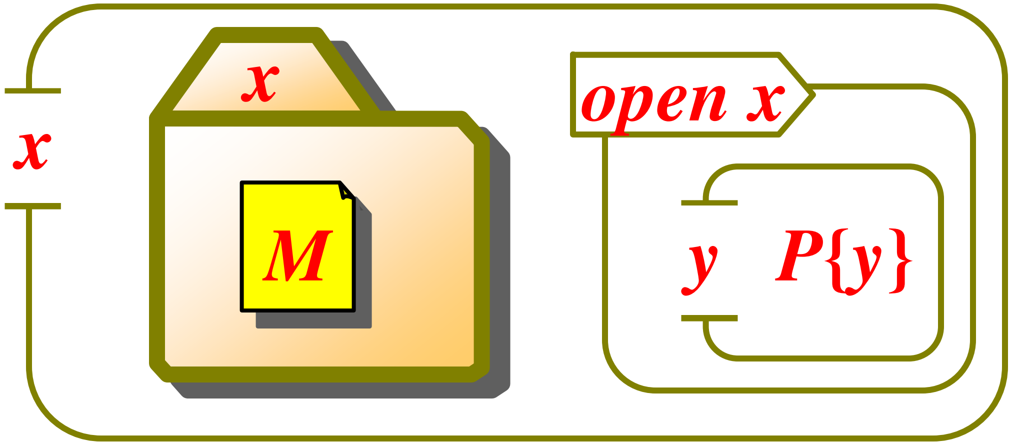


Example: A Distiller Server

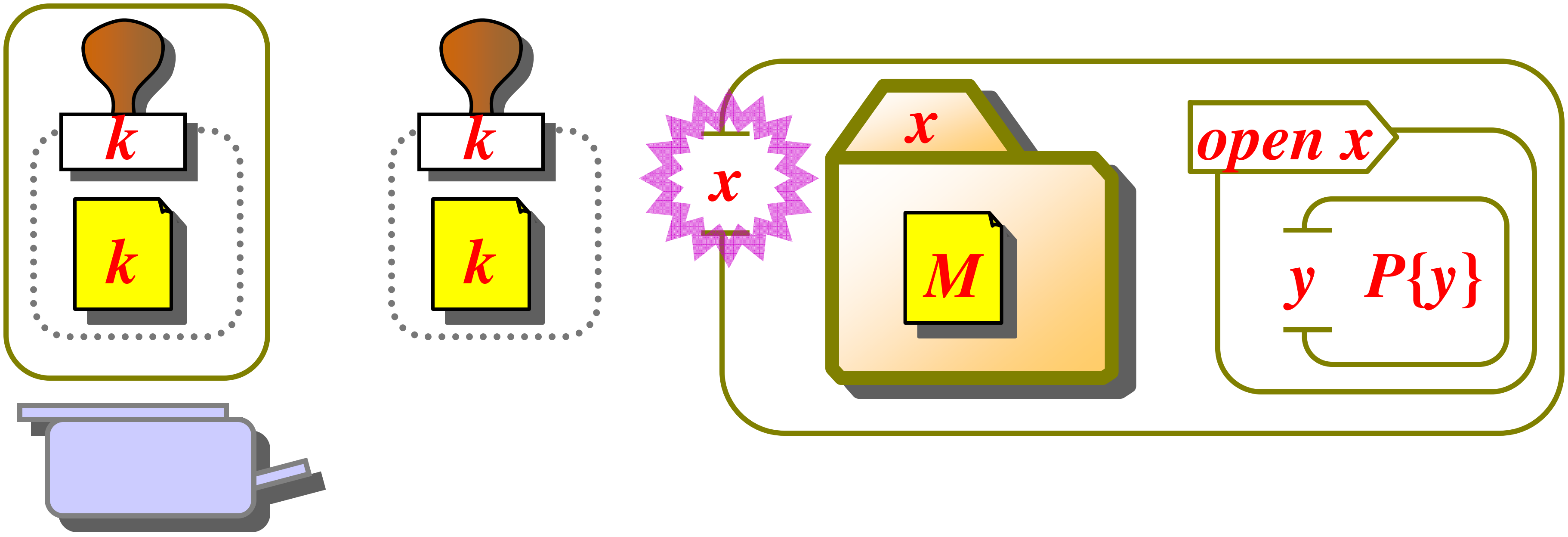


Example: Keys



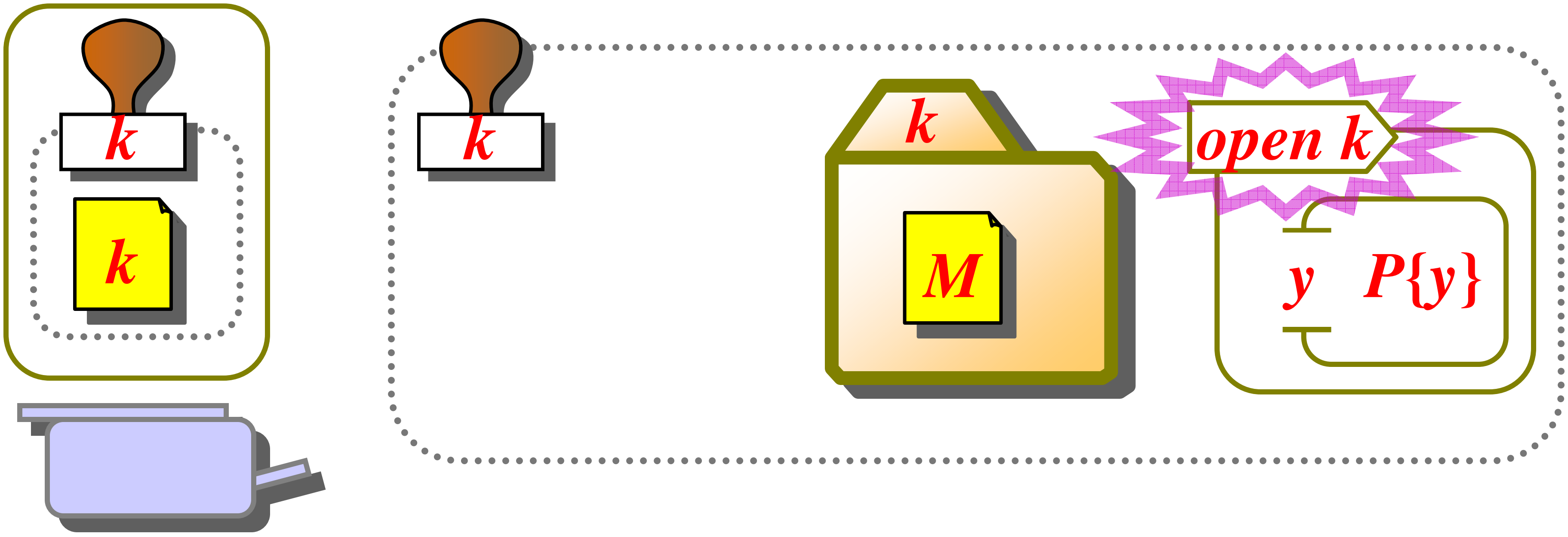


Example: Keys



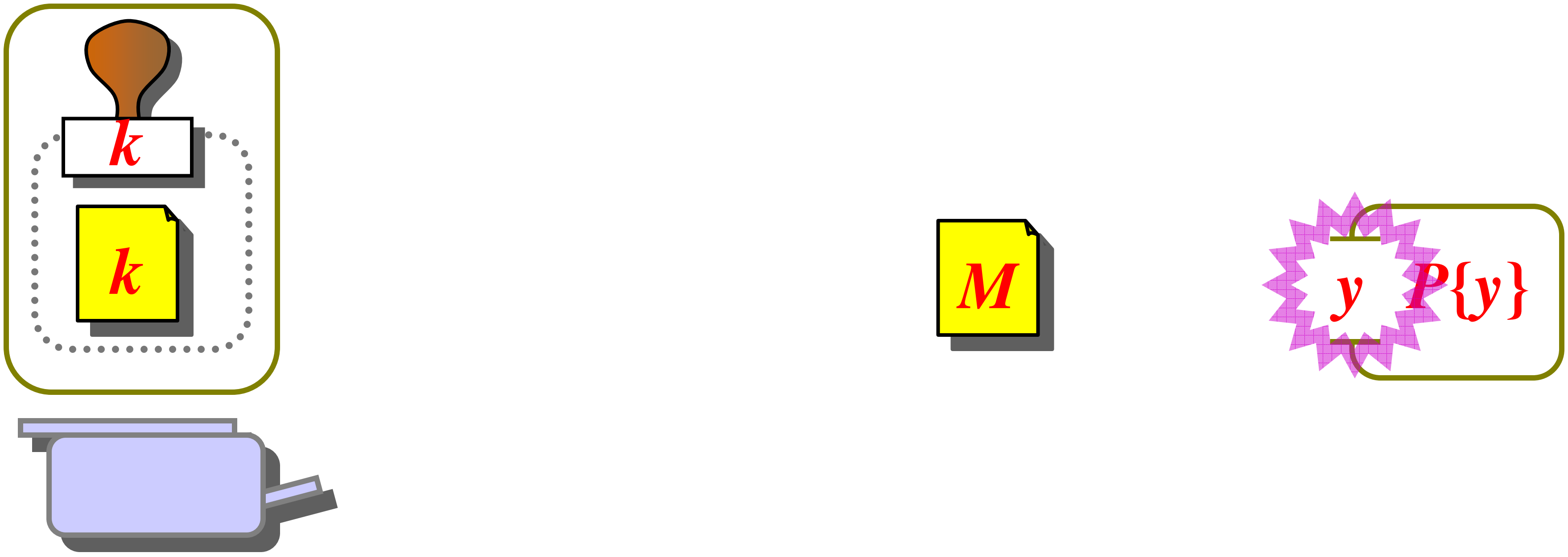
- Generate a fresh key *k*.

Example: Keys



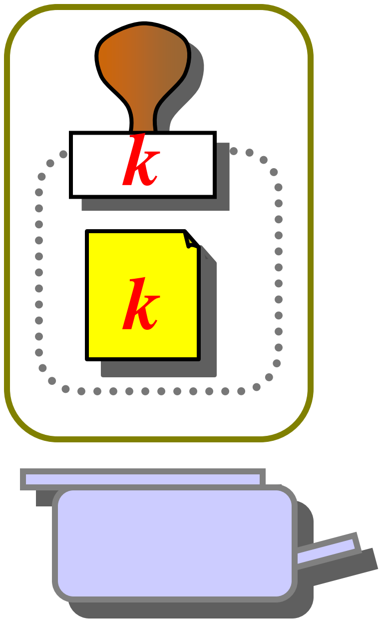
- Generate a fresh key *k*.
- Encrypt *M* under *k*.

Example: Keys



- Generate a fresh key *k*.
- Encrypt *M* under *k*.
- Decrypt via *open k* …

Example: Keys



$P\{M\}$

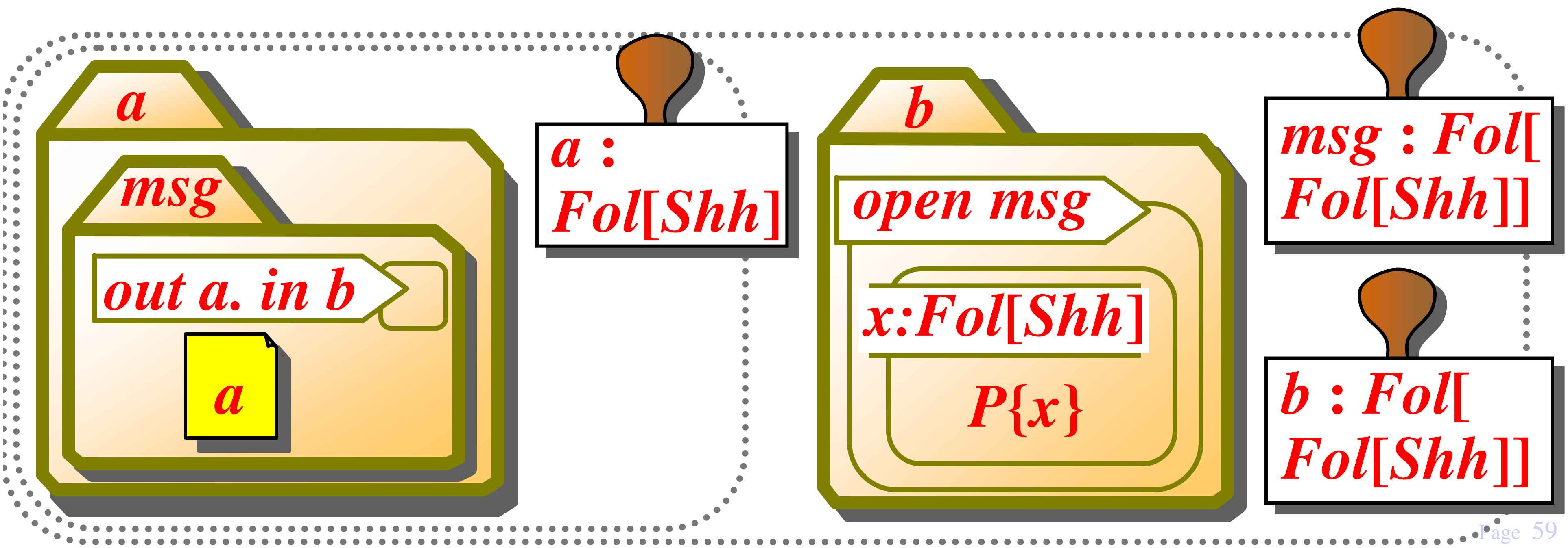
- Generate a fresh key *k*.
- Encrypt *M* under *k*.
- Decrypt via *open k* and read *M*.

Remarks

- The folder calculus is Turing-complete (even without the I/O operations),
- It's highly concurrent, with synchronization primitives.
- A type system can be used to make sure that each input reads only messages of the appropriate type.
  - The type of a file is associated with the name of the folder that contains it. All the files in a folder must have the same type.
  - Subfolders of a given folder may contain files of different types.
  - So we have a heterogeneous data hierarchy, but with well-typed I/O.

A Flexibly-Typed "File System"

- *n* : *Fol*[*T*] means that *n* is a name for folders that can contain (or exchange) files (or messages) of type *T*. All folders named *n* can contain only *T* files/messages.
- Nothing is said about the subfolders of folders of name *n*: they can have any name and type (and can come and go).
- Mobility is totally unconstrained by this type system.



Expressiveness

- This seems simple-minded, but it is very expressive:
- Channel types: $Ch[T] \triangleq Fol[T]\times Fol[T]$
  A channel name is a pair of folder names ("buffer" folders and "packet" folders respectively).
- Function types: $A\rightarrow B \triangleq Ch[A\times Ch[B]]$
  A function from A to B is (used through) a channel to which we give an argument of type A and the name of a channel in which to deposit the result of type B.
- Agent types:
  An agent is a mobile process that performs well-typed I/O on channels at different locations.

Mobility Types

- The previous type system can be refined with additional information, in order to constrain mobility.
- A folder may be declared immobile (cannot move on its own), or locked (cannot be opened). This information can be tracked statically.
- Application: make sure, at compile-time or a load-time, that applets cannot move around, or that dangerous packages cannot be accidentally opened.

Mobility Group Types

- $G[T]$ (generalizing $Fol[T]$) is the type of the names of group G, which name folders that can contain messages of type T.
  - Assert that folders of group G can enter/exit only folders of group $G_1 ... G_n$ (generalizing sandboxing).
  - Assert that a process can open only packages of group G (generalizing locking).
  - New groups can be dynamically created; e.g.: private groups. This has the effect of statically preventing the accidental escape of capabilities to third parties.
- Application: enforcement, at compile-time or load-time, of "mobility policies" and "assimilation policies" for applets.

A Spatial Logic

- We have been looking for ways to express properties of mobile computations and of mobility protocols. E.g.:
  - "Here today, gone tomorrow."
  - "Eventually the agent crosses the firewall."
  - "Every agent carries a suitcase."
  - "Somewhere there is a virus."
  - "There is always at most one folder called n here."
- Solution: devise a process logic that can talk about space (as well as time).
- This can be seen as a generalization of the mobility types to less easily checkable (but more interesting) mobility properties.

Examples of Formulas

- The folder calculus has a spatial structure given by the nesting of folder: we want a logic that can talk about that structure:

Formulas	
$\mathbf{0}$	(there is nothing here)
$n[\mathcal{A}]$	(there is one thing here)
$\mathcal{A} \mid \mathcal{B}$	(there are two things here)
$\mathbf{T}$	(there is anything you want here)
$✧\mathcal{A}$	(somewhere down here $\mathcal{A}$ holds)
$\diamond\mathcal{A}$	(sometime in the future $\mathcal{A}$ may hold)
$\mathcal{A} \triangleright \mathcal{B}$	($\mathcal{B}$ is satisfied even under an $\mathcal{A}$ attack)

- Ex., *p parents q*: $✧(p[q[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T})$
- Ex., *m may exit n*: $n[✧m[\mathbf{T}]] \wedge \diamond(n[\mathbf{0}] \mid m[\mathbf{T}])$

Satisfaction

- The logic is defined explicitly via a satisfaction relation:

$$P \models \mathcal{A}$$

  meaning that the configuration (model) P satisfies the formula $\mathcal{A}$.

- For a subset of this relation we have a model-checking algorithm (i.e., a decision procedure).
- Applications:
  - compile-time or load-time checking of interesting properties of mobile code.
  - Enforcement of mobility and/or security policies of mobile code. Easier properties may be checked by model-checking, harder ones by theorem-proving or theorem-checking (e.g., proof-carrying code).

From Calculi to Languages

- The ambient calculus is a minimal formalism designed for theoretical study. As such, it is not a "programming language".
- Still, the ambient calculus is designed to match fundamental WAN characteristics.
- By building languages on top of a well-understood WAN semantics, we can be confident that languages will embody the intended semantics.
- We now discuss how ambient characteristics might look like when extrapolated to programming languages.

WAN Observable Phenomena

- Physical Locations
  - Observable because of the speed of light limit
  - Preclude instantaneous actions
  - Require mobile code
- Virtual Locations
  - Observable because of administrative domains
  - Preclude unfettered actions
  - Require security model and disconnected operation

- Variable Connectivity
  - Observable because of free-will actions, physical mobility
  - Precludes purely static networks
  - Requires bandwidth adaptability
- Failures
  - Unobservable because of asynchrony, domain walls
  - Preclude reliance on others
  - Require blocking behavior, transaction model

Wide Area Languages

- Languages for Wide Area Networks:
- WAN-sound
  - No action-at-a-distance assumption
  - No continued connectivity assumption
  - No security bypasses
- WAN-complete
  - Able to emulate surfer/roamer behavior
- Some steps towards Wide Area Languages:
  - Ambient Calculus (with Andy Gordon)
  - Service Combinators (with Rowan Davies)
  - … various B2B languages/systems being proposed

Summary of WAL Features

- No “hard” pointers.
  - Remote references are URLs, symbolic links, or such.
- Migration/Transportation
  - Thread migration.
  - Data migration.
  - Whole-application migration.
- Dynamic linking.
  - A missing library or plug-in may suddenly show up.
- Patient communication.
  - Blocking/exactly-once semantics.
- Built-in security primitives.

Current Status

- Concepts
  - An informal paper describing wide-area computation, the Folder Calculus, and ideas for wide-area languages.
- Semantics (with Andy Gordon)
  - Semantics of the basic Ambient Calculus.
  - Techniques for proving equational properties of Ambients.
- Type Systems (with Andy Gordon and Giorgio Ghelli)
  - A type systems for Ambients, regulating communication.
  - Type systems for constraining the diffusion of capabilities and for regulating mobility.
- Logics (with Andy Gordon)
  - Describing spatial and temporal Ambient properties.

- Implementation (Multiple strategies)
  - A Java applet implementation of the Ambient Calculus, and a tech report about its thread synchronization algorithm.
  - (With Leaf Petersen) Stopping, linearizing, and restarting Ambient configurations.
  - (With Mads Torgesen) Design and implementation of a "large-scale" Ambient-based programming language.
  - (Simon Peyton Jones) Experiments in implementing Ambient primitives in Concurrent Haskell.
  - (Cédric Fournet, Alan Schmitt - INRIA) A distributed implementation of Ambients in JoCaml.

Conclusions

- The notion of named, hierarchical, mobile entities captures the structure and properties of computation on wide-area networks.
- The ambient calculus (exemplified by the folder calculus) formalizes these notions simply and powerfully.
  - It is no more complex than common process calculi.
  - It supports reasoning about mobility and security.
- We believe we have a solid basis for envisioning new programming methodologies, libraries, and languages for wide-area computation.