Wide Area Computation

Luca Cardelli

Microsoft Research

Joint work with Andrew D. Gordon
and occasional others

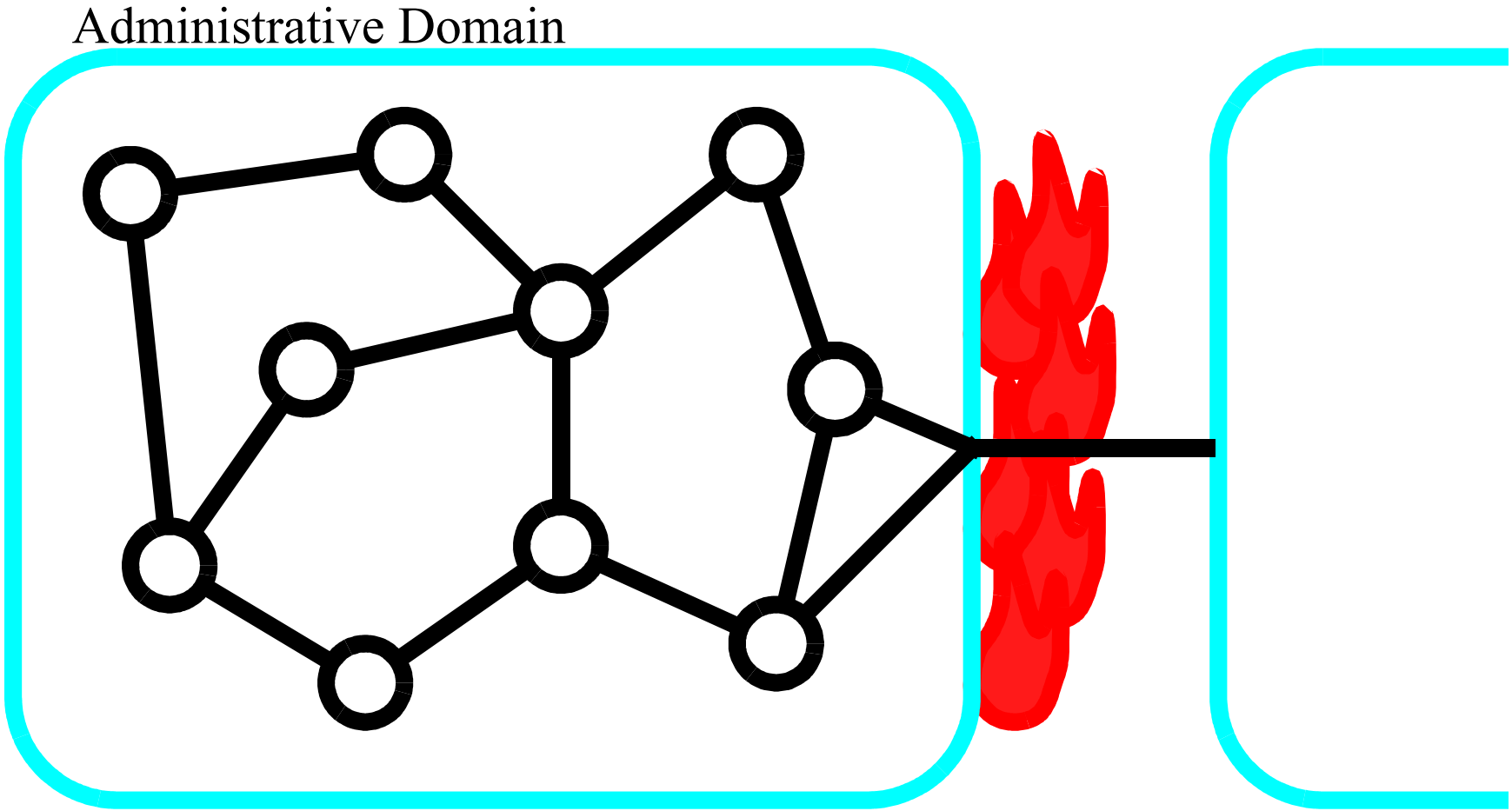
Wide Area Computation

Three Mental Pictures

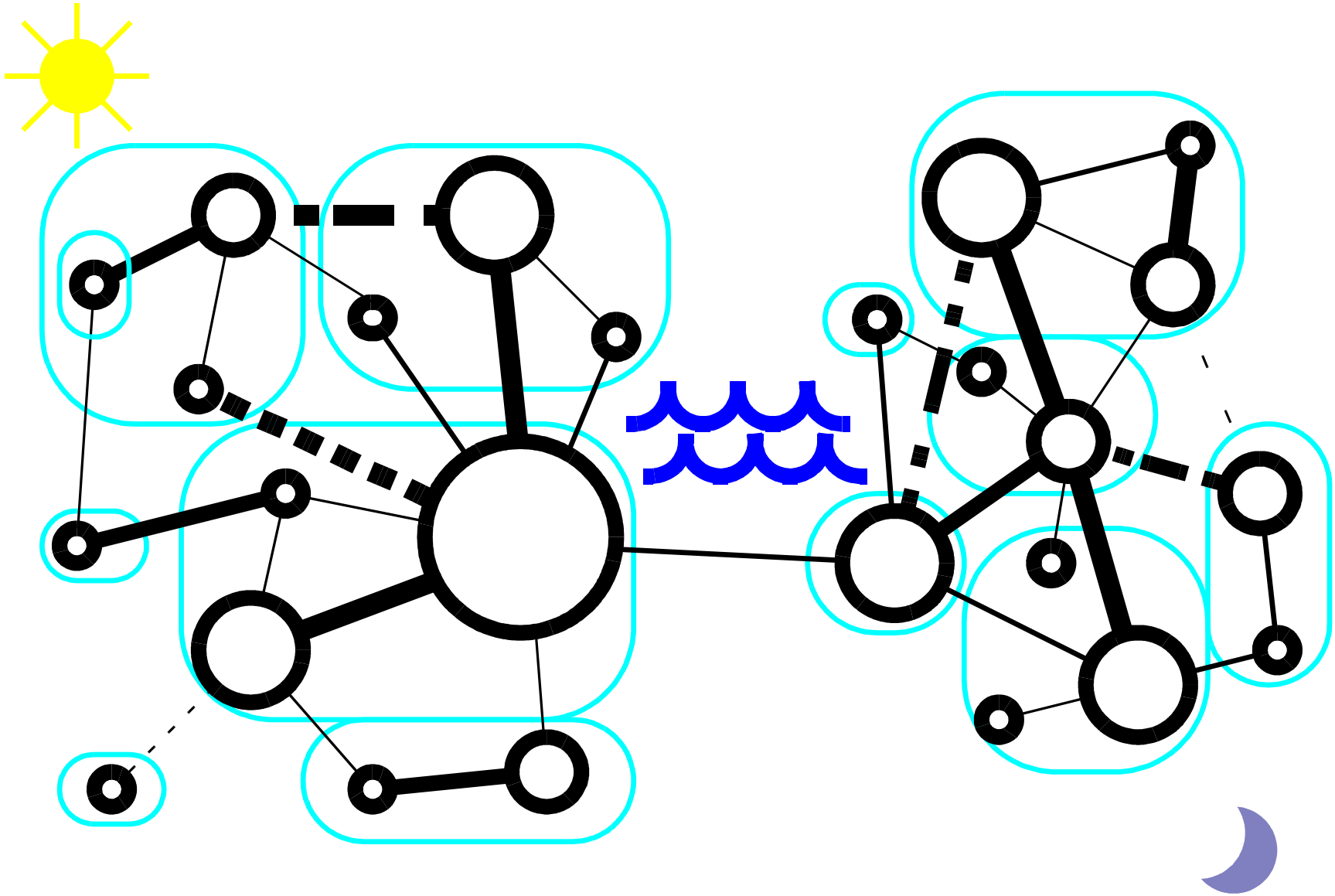
Three views of computation:

- Local area networks
- Wide area networks
- Mobile networks

LANs and (Traditional) Distributed Computing



The Web



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***.

A WAN is not a big LAN

- To emulate a LAN on top of a WAN we would have to:
  - ***(A) Hide virtual locations***. By semi-transparent security. But is it possible to guarantee the integrity of mobile code?
  - ***(B) Hide physical locations.*** Cannot "hide" the speed of light, other than by slowing down the whole network.
  - ***(C) Hide bandwidth fluctuations***. Service guarantees eliminate bandwidth fluctuations, but introduce access failures.
  - ***(D) Reveal failures.*** Impossible in principle, since the Web is an asynchronous network.
- Hard problems: (A) may be unsolvable for mobile code; (B) is only solvable (in full) by introducing unacceptable delays; (C) can be solved in a way that reduces it to (D); (D) is unsolvable in principle, while probabilistic solutions run into (B).

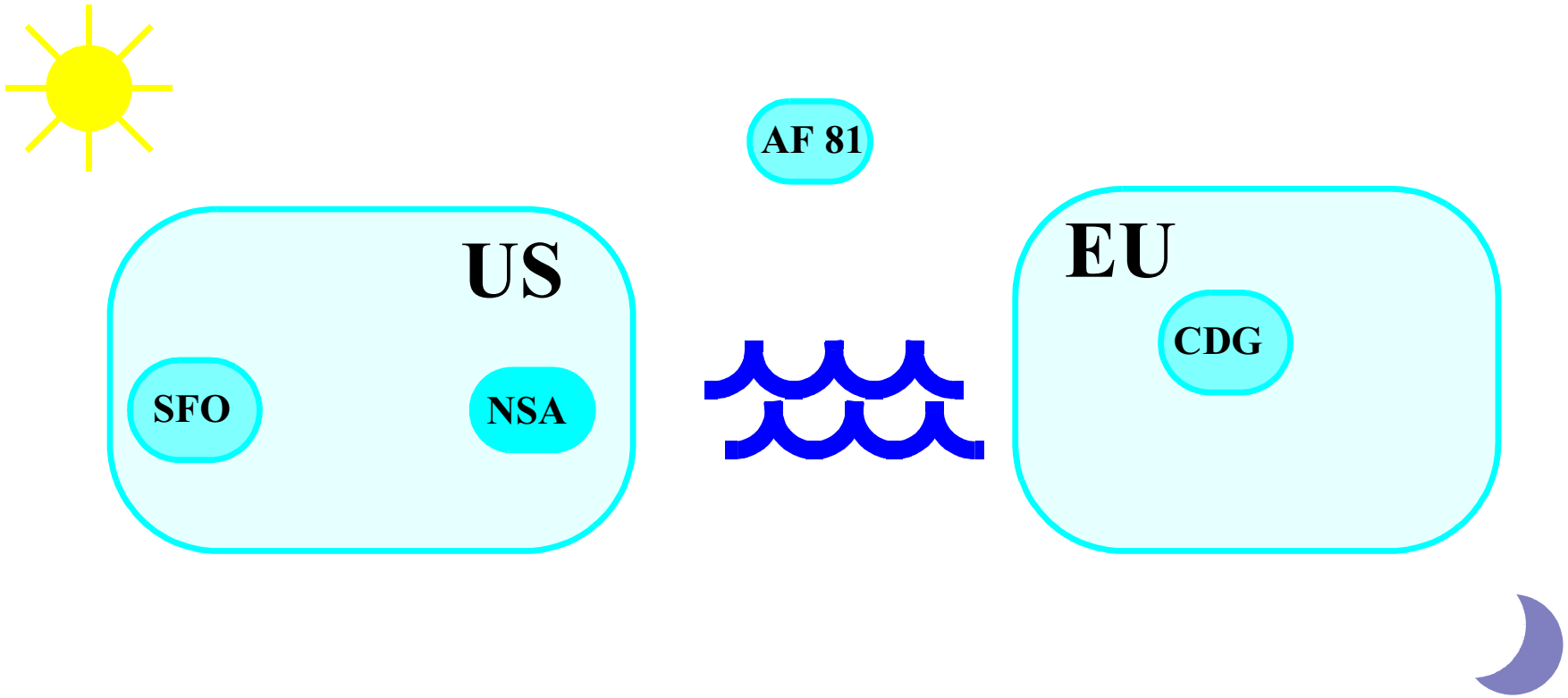
Observables

- WAN *observables* are different (and not reducible to) LAN observables.
- Observables determine programming constructs, and therefore influence programs and programming languages.
- We need a complete set of programming constructs that can detect and react to the available observables and, of course, we do not want programming constructs that attempt to detect or react to non-observables.

Mobile Computation

- Mobile computation can cope with the observables characteristic of a wide-area network such as the Web.
  - ***Virtual locations***. Trust mechanisms to cross virtual barriers.
  - ***Physical locations***. Mobility to optimize placement.
  - ***Bandwidth fluctuations***. Mobility to split applications and establish optimized communication protocols.
  - ***Failures***. Running around or away from failures.

Mobile Computing



- Mobile devices also move computations. In this sense, we cannot avoid the issues raised by mobile computation.

Mobility Postulates

- Separate locations exist. They may be difficult to reach.
- Since different locations have different properties, both people and programs (and sublocations!) will want to move between them.
- Barriers to mobility will be erected to preserve certain properties of certain locations.
- Some people and some programs will still need to cross those barriers.

This is the situation wide area computing has to cope with.

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

Modeling Wide Area Computation

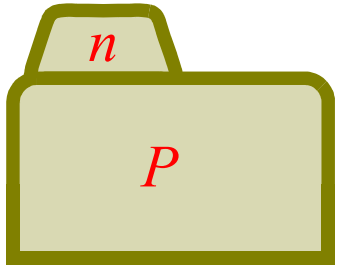
- Locality: **Barrier topology**.
  - *Cf.* failure semantics, named processes.
- Process mobility: **Barrier crossing**.
  - *Cf.* name passing (π), process passing (CHOCS).
- Security: **(In)ability to cross barriers**.
  - *Cf.* cryptography (Spi), flow control (SLAM)
- Interaction: **Shared ether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

FOLDER CALCULUS

The Folder Calculus

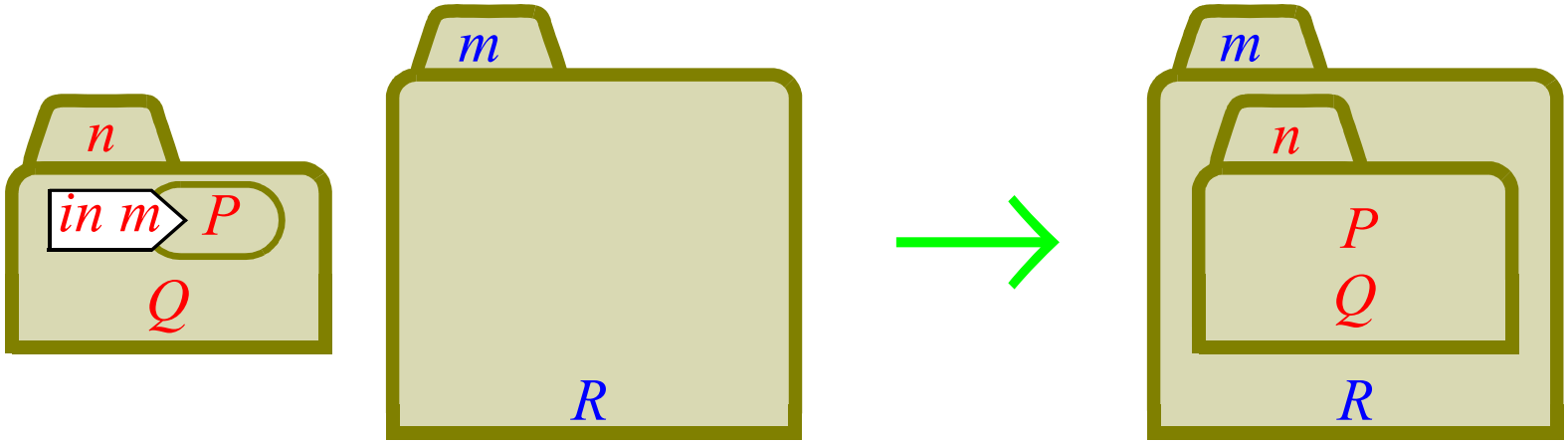
- A graphical office metaphor to explain the ambient calculus.
- A precise metaphor, isomorphic to the formal ambient calculus.
- Based on wide-area computation principles: locality, mobility, nested domains, asynchronous communication, authentication.

Folders (Nested Domains)

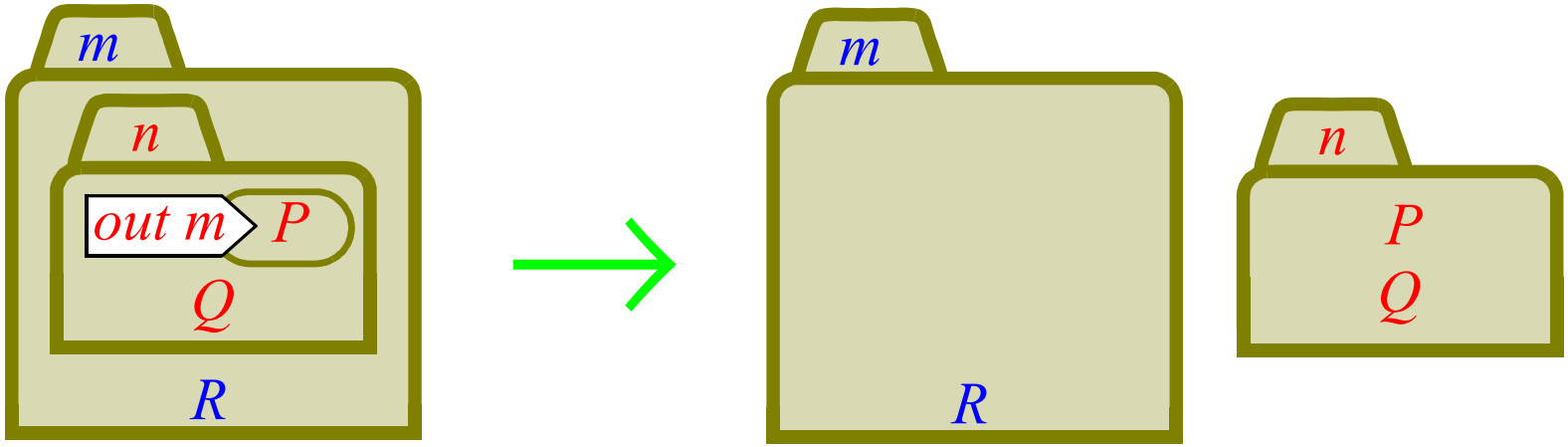


- A folder name *n*.
- Active contents *P*:
  - hierarchical data and “gremlins”.
  - computational primitives for mobility and communication.

Enter Reduction (Mobility)

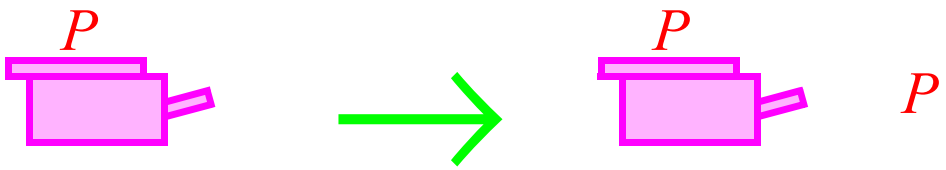


Exit Reduction (Mobility)

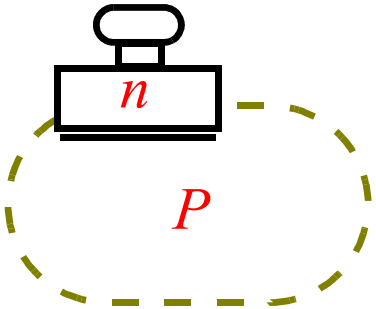


Open Reduction (Assimilation)

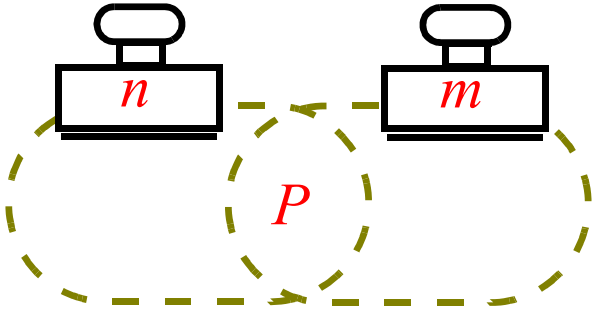
Copy Reduction (Iteration/Recursion)



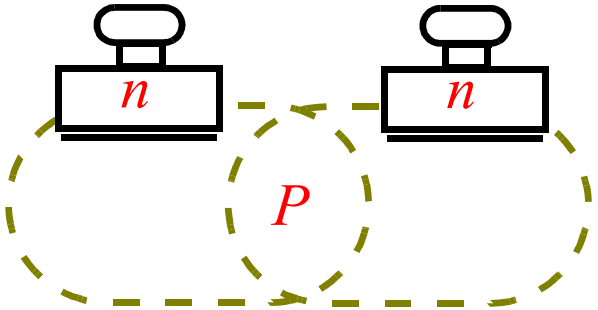
Rubber Stamps (Authentication)



- Give authenticity to folders.
- Copiers are unable to accurately duplicate rubber stamps.



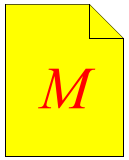
allowed



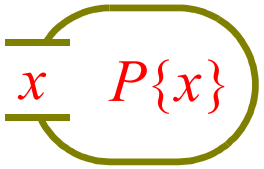
forbidden

Post-It Notes (Local Communication)

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



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

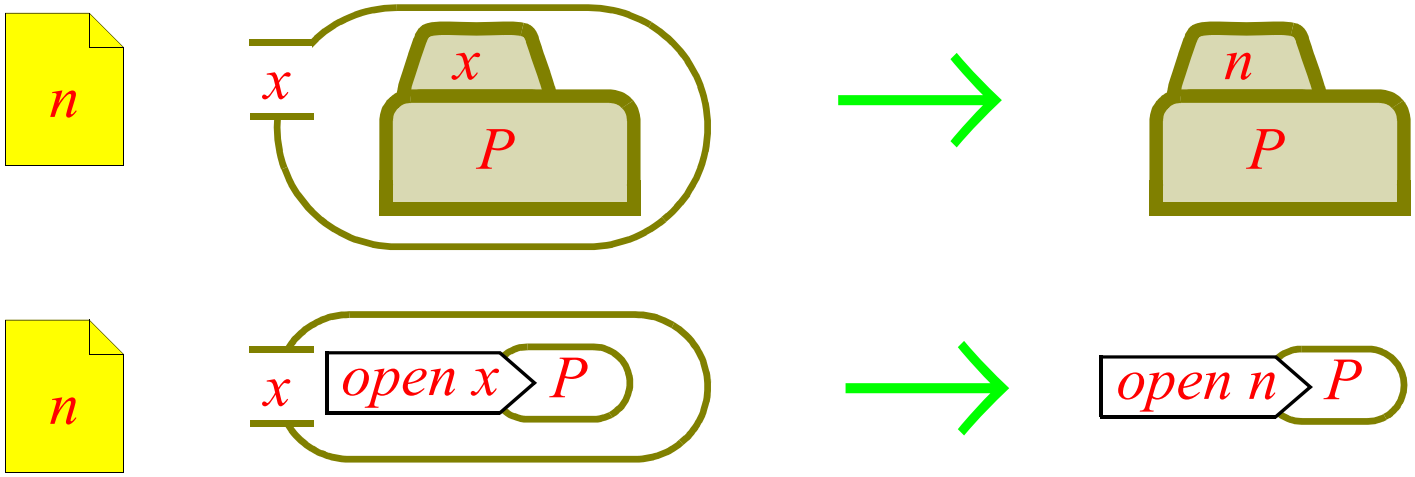


- Read reduction

Messages (Names or Capabilities)

A message *M* can be either:

- The name of a folder (danger: spoofing, killing):

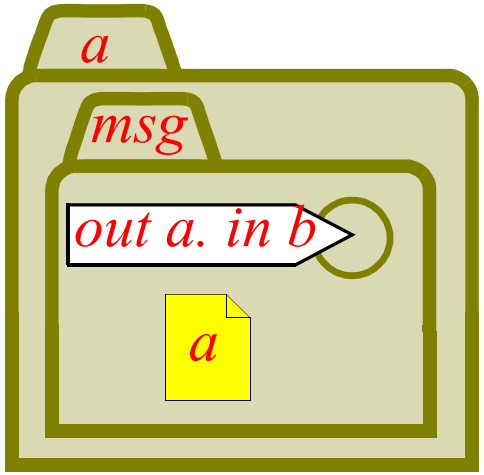


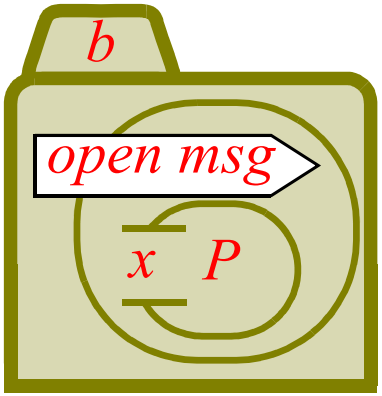
- A capability (no danger of recovering the name):

Leaves of the Syntax

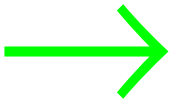
Inactive gremlin

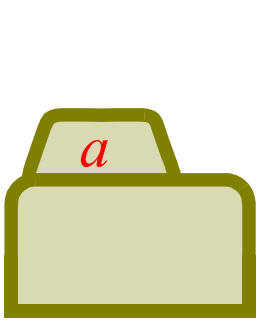
Example: Message from A to B

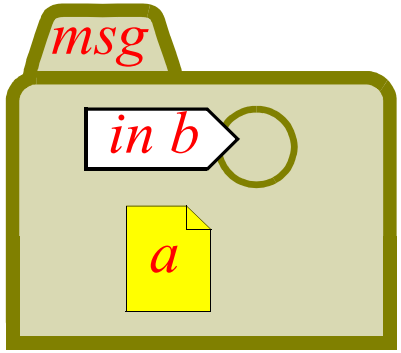


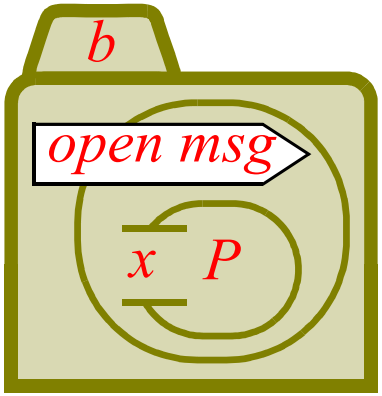


Exit

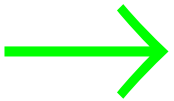


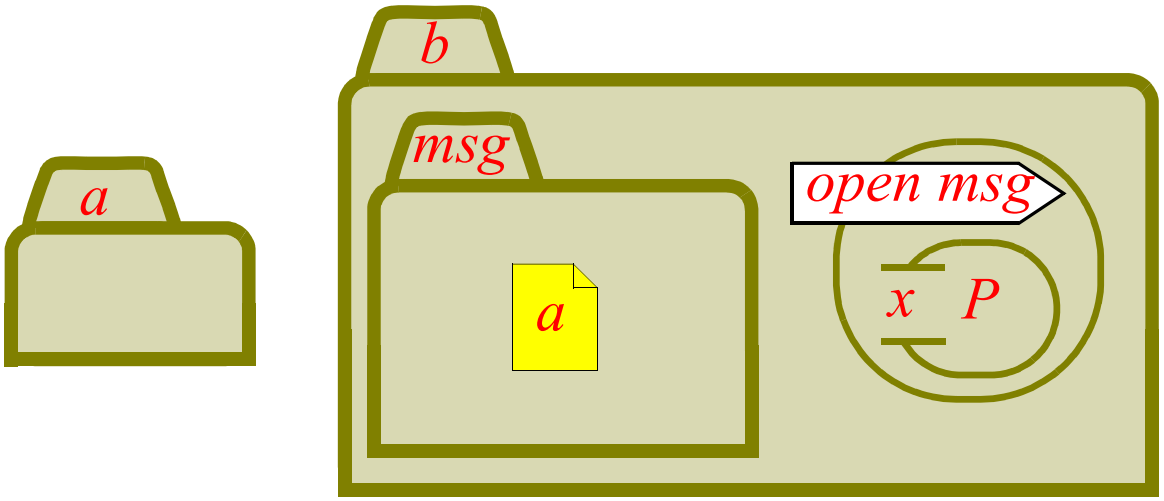




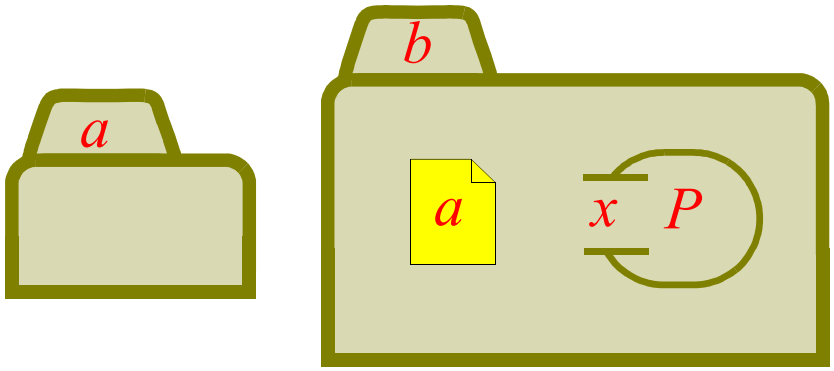


Enter

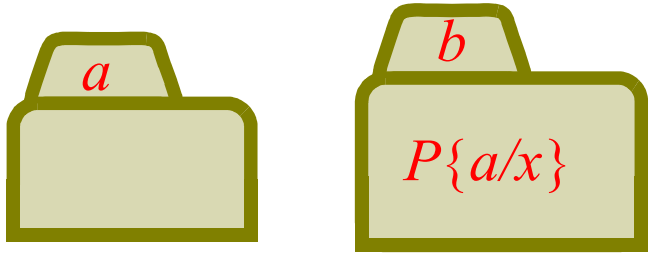




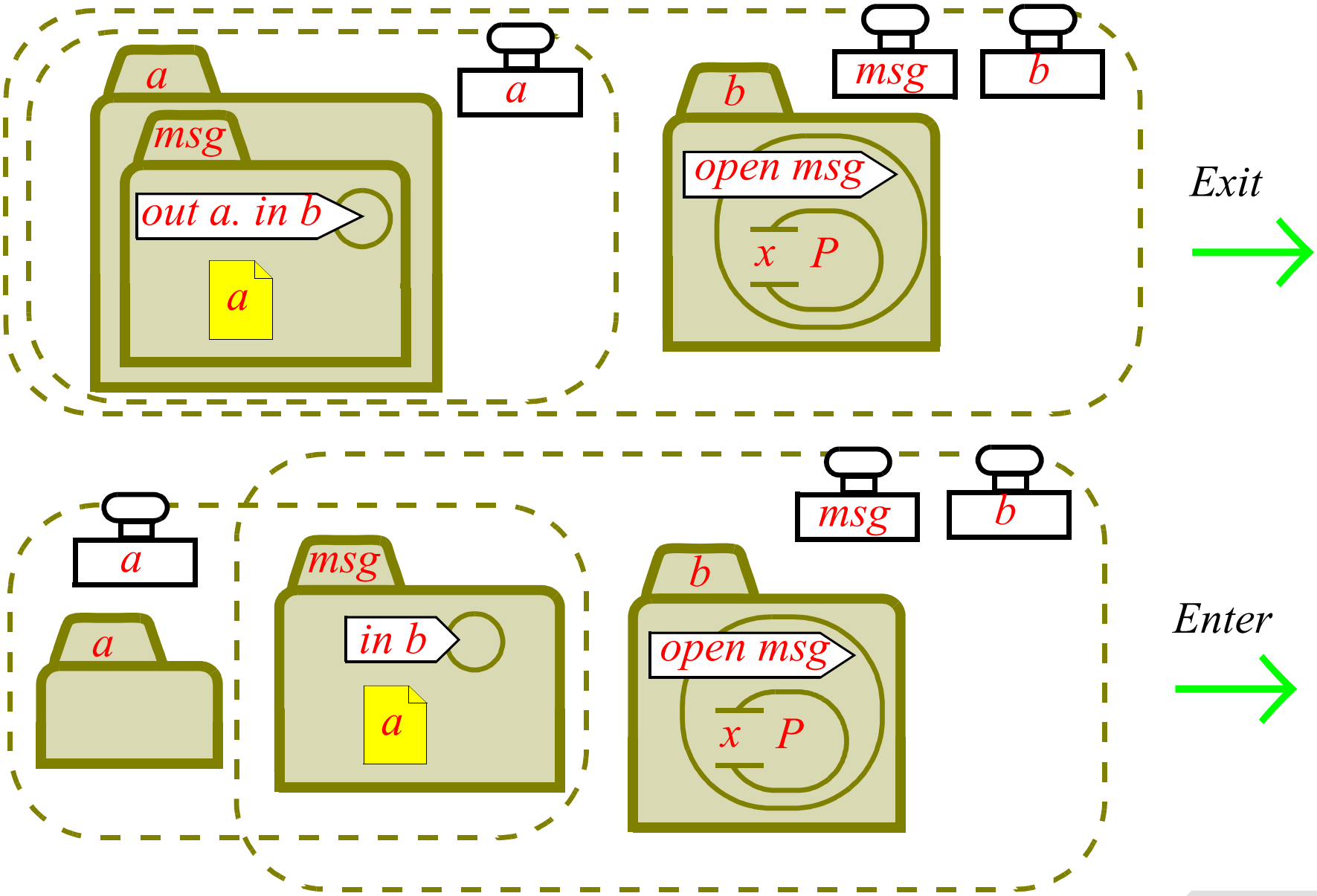
Open

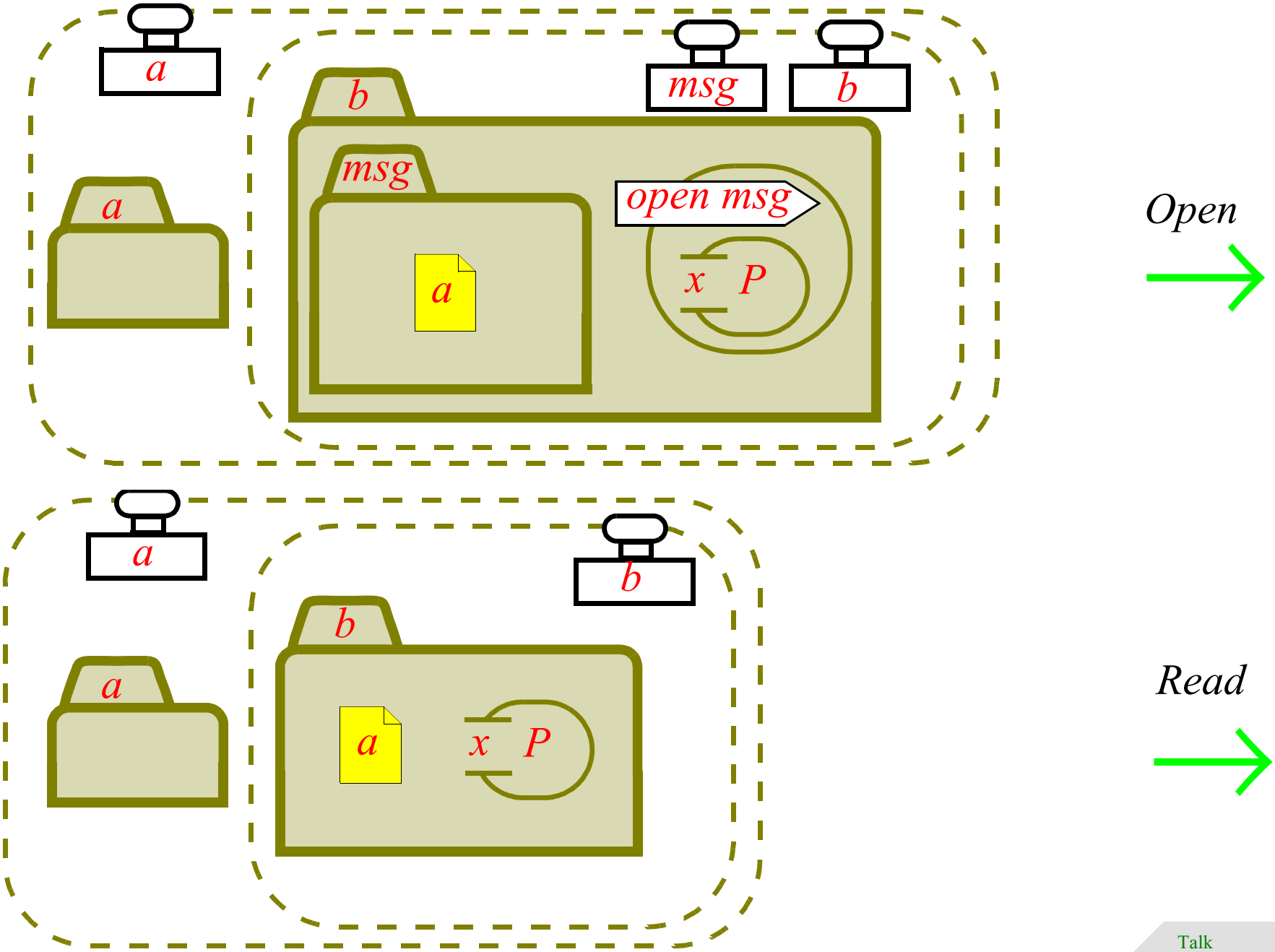


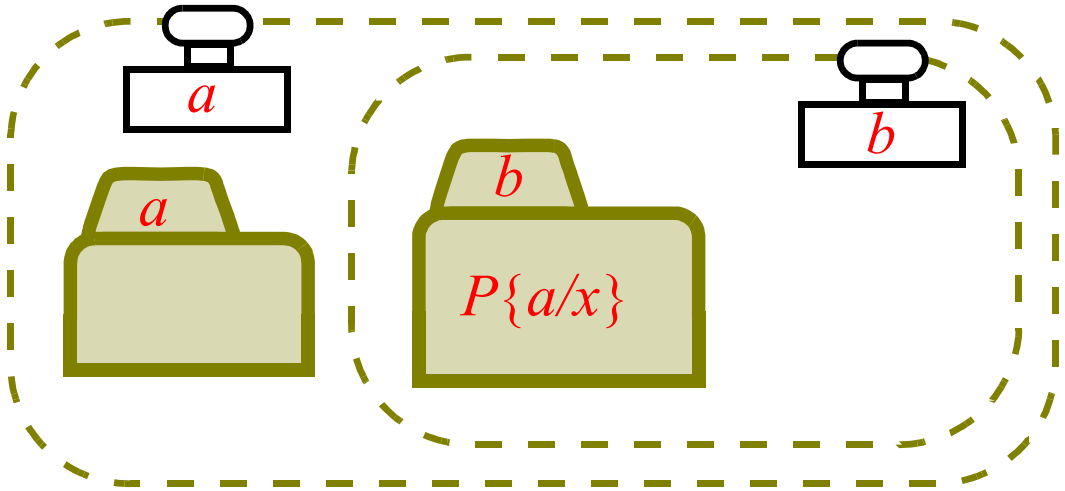
Read


a
b
P{a/x}

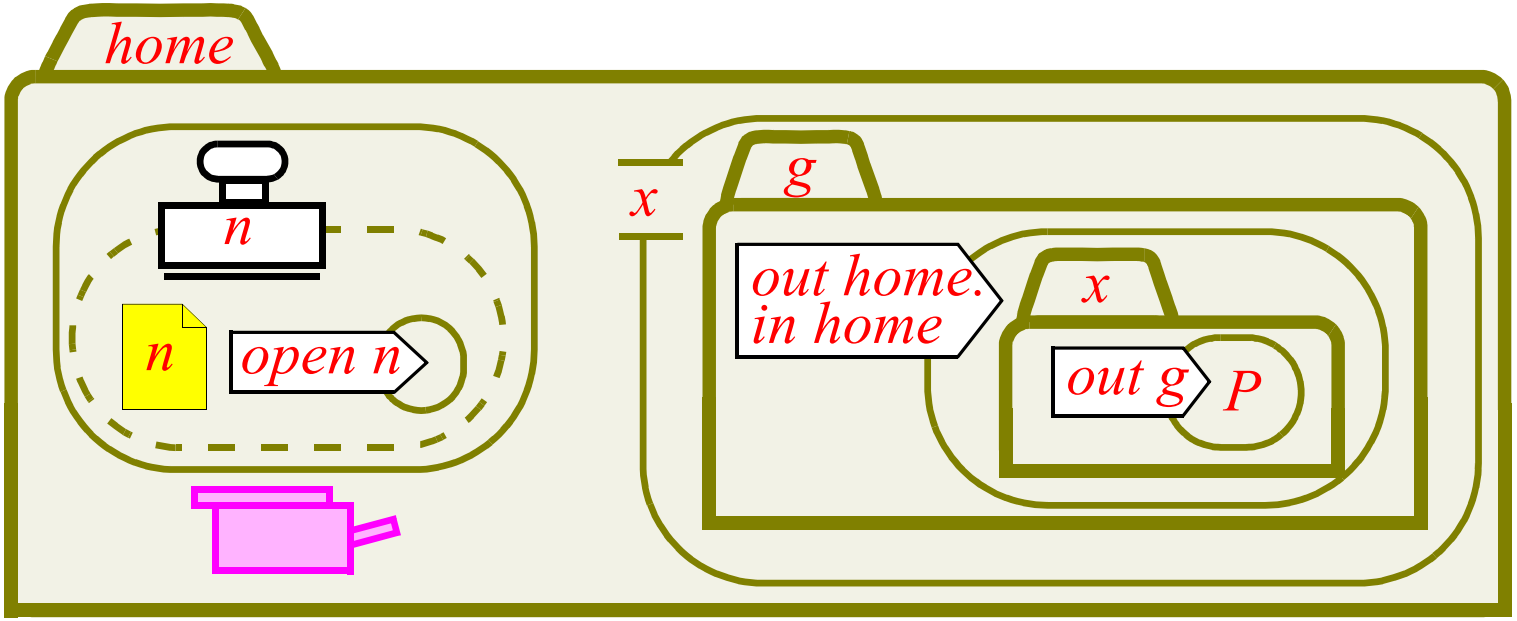
Same Example, with scoping



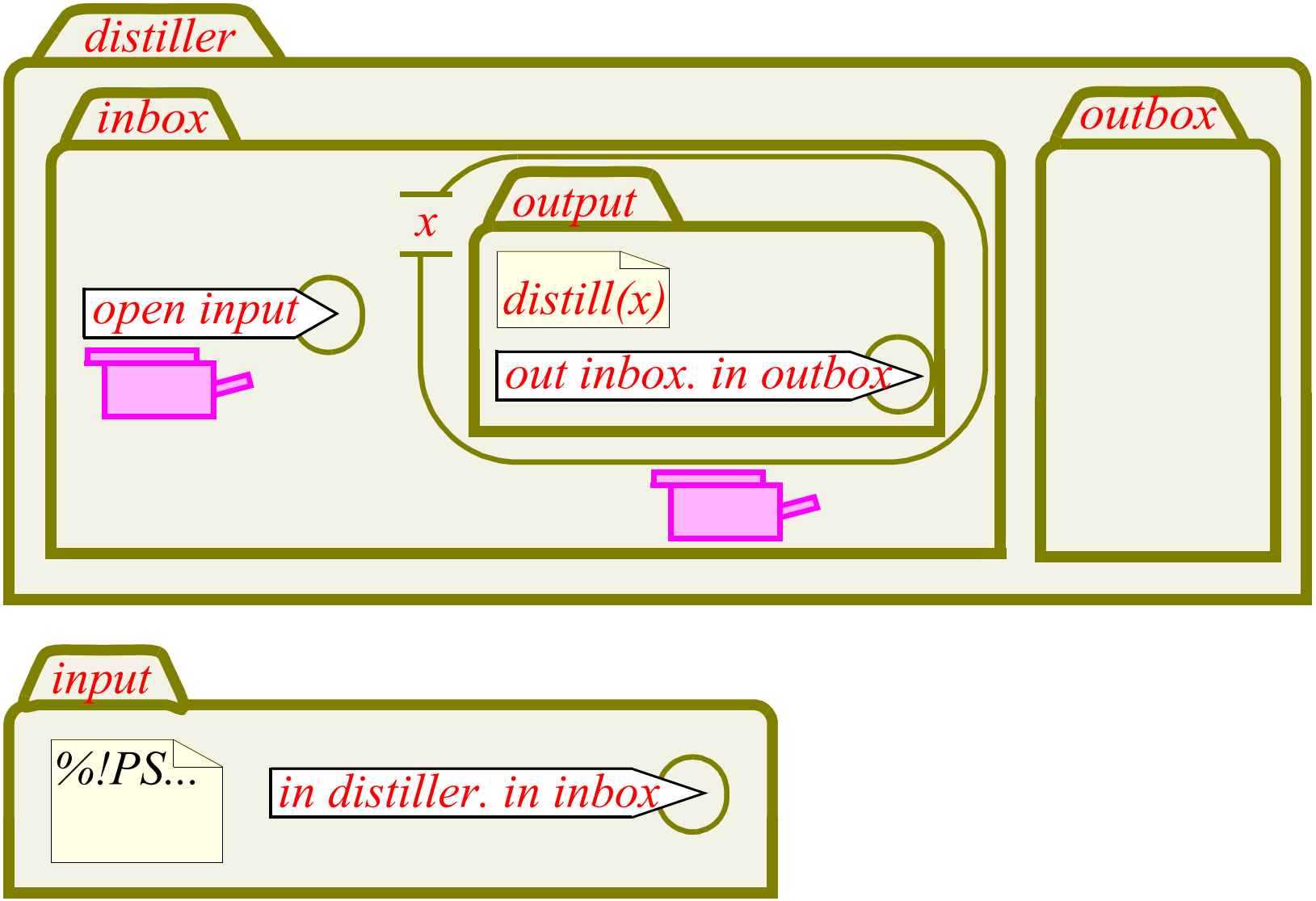

a
b
msg
b
msg
a
a
open msg
x P
Open
a
b
b
a
a
x P
Read


a
a
b
b
P{a/x}

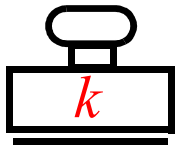
Example: Agent Authentication



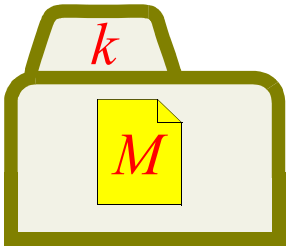
Example: A Distiller Server



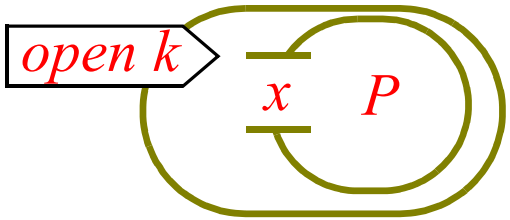
Example: Keys



generation of a fresh key k



encryption:
plaintext M inside a k-envelope

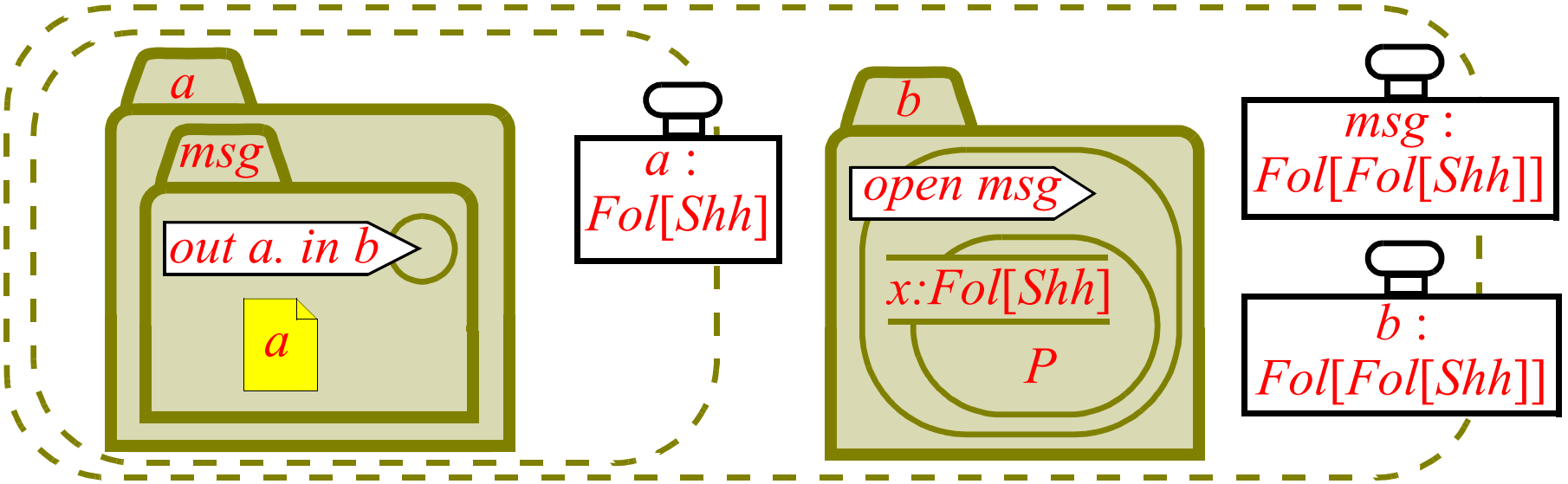


decryption:
opening a k-envelope and reading the contents

TYPE SYSTEMS FOR MOBILITY

A Flexibly-Typed "File System"

- $n : Fol[T]$ means that n is a name for folders that can contain (or exhange) 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 any type (and can come and go).
- Mobility is totally unconstrained by this type system.



Expressiveness

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" foldes 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 foders 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 (generalazing 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.

LOGICS FOR MOBILITY

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).

AMBIENT CALCULUS EXPRESSIVENESS

Expressiveness: Old Concepts

- Synchronization and communication mechanisms.
- Turing machines. (Natural encoding, no I/O required.)
- Arithmetic. (Tricky, no I/O required.)
- Data structures.
- π-calculus. (Easy, channels are ambients.)
- λ-calculus. (Hard, different than encoding λ in π.)
- Spi-calculus concepts.

Expressiveness: New Concepts

- Named machines and services on complex networks.
- Agents, applets, RPC.
- Encrypted data and firewalls.
- Data packets, routing, active networks.
- Dynamically linked libraries, plug-ins.
- Mobile devices.
- Public transportation.

Expressiveness: New Challenges

- The combination of mobility and security in the same formal framework is novel and intriguing.

  E.g., we can represent both mobility and security aspects of “crossing a firewall”.

- The combination of mobility and local communication raises questions about suitable synchronization models and programming constructs.

WAN 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)

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.

Conclusions: Ambients

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.