Abstractions for Mobile Computation

Luca Cardelli

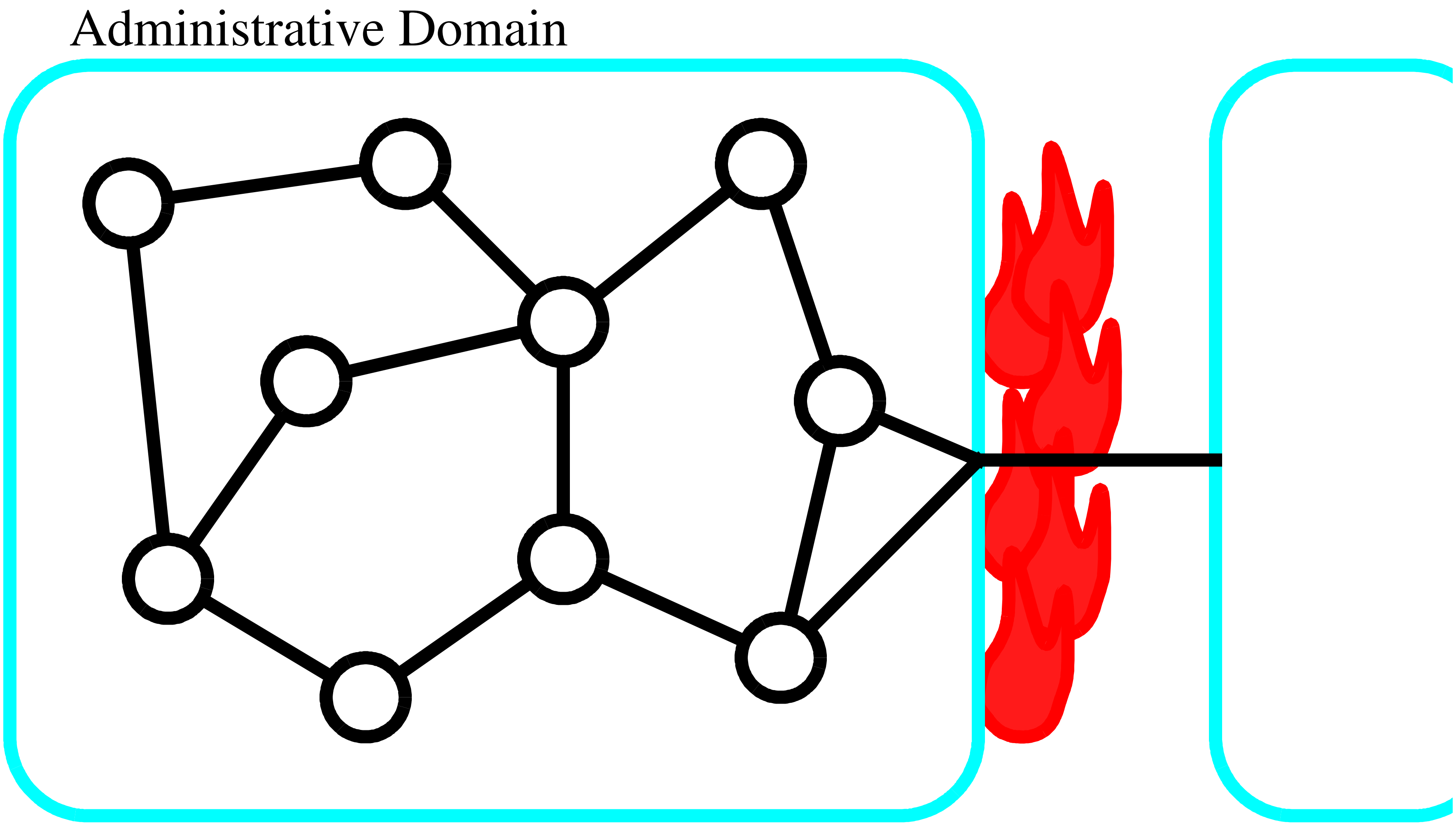
(with Andrew D. Gordon)

Outline

- Mobility in the real and virtual world.
  - ~ Informal review of what's out there.
- Modeling mobility.
  - ~ Previous work.
  - ~ The ambient calculus.
  - ~ Examples.
- Applications.
  - ~ Verification of combined security and mobility properties.
  - ~ New mobility libraries/languages.

Three Mental Pictures

(Traditional) Distributed Computing



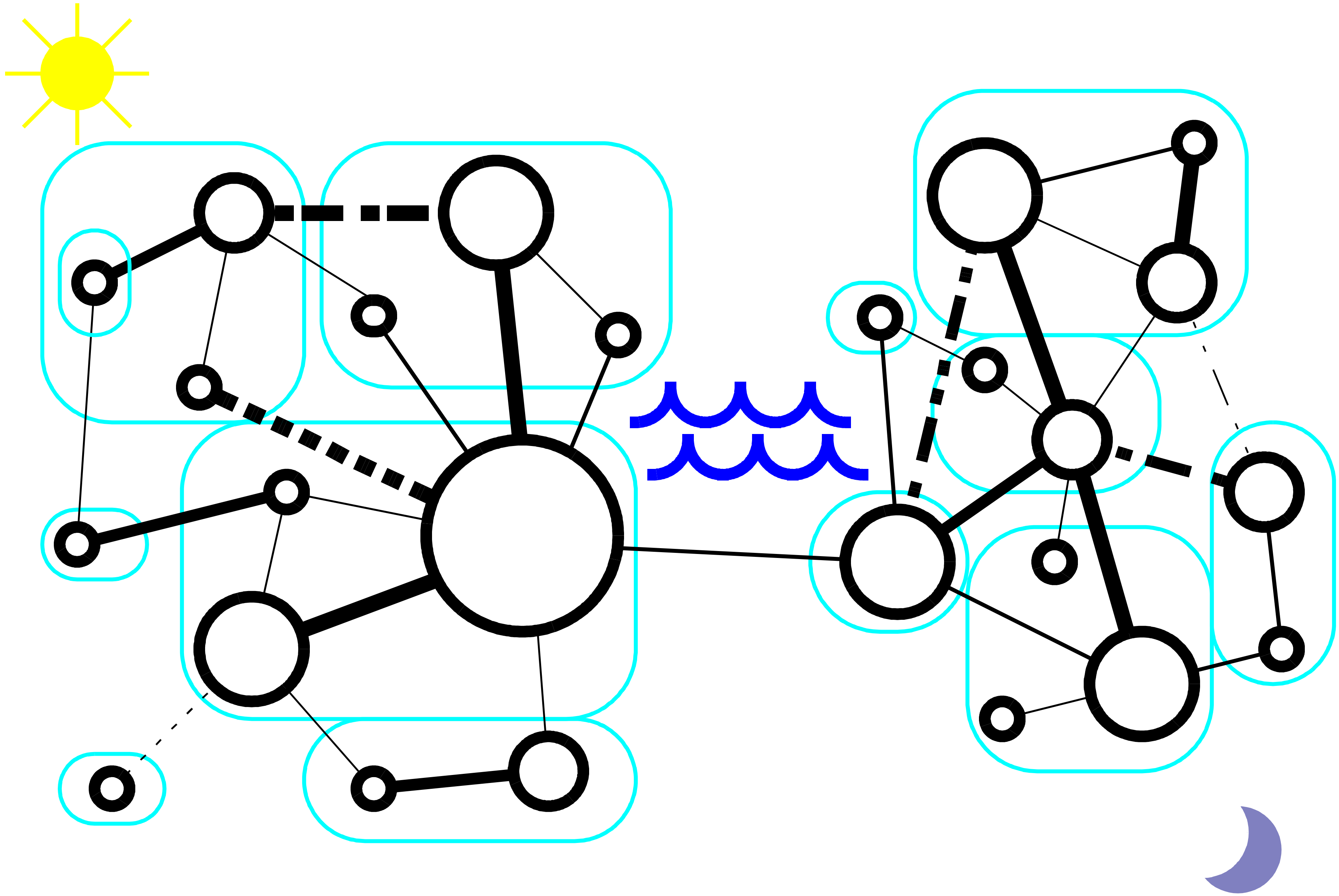
Immobility
(Traditional Distributed Object Systems)

- RPC/RMI.
  (CORBA, OLE, Modula-3 Network Objects, Java RMI.)
- Control mobility, data mobility, link mobility.
- No code mobility, no thread/process mobility.
- Static and often trivial topology (everything 1 logical step apart).

Virtual Mobility
(Pre-Web Software Systems)

- Tcl. (Code mobility.)
- Telescript. (Agent mobility.)
- Obliq. (Closure mobility.)

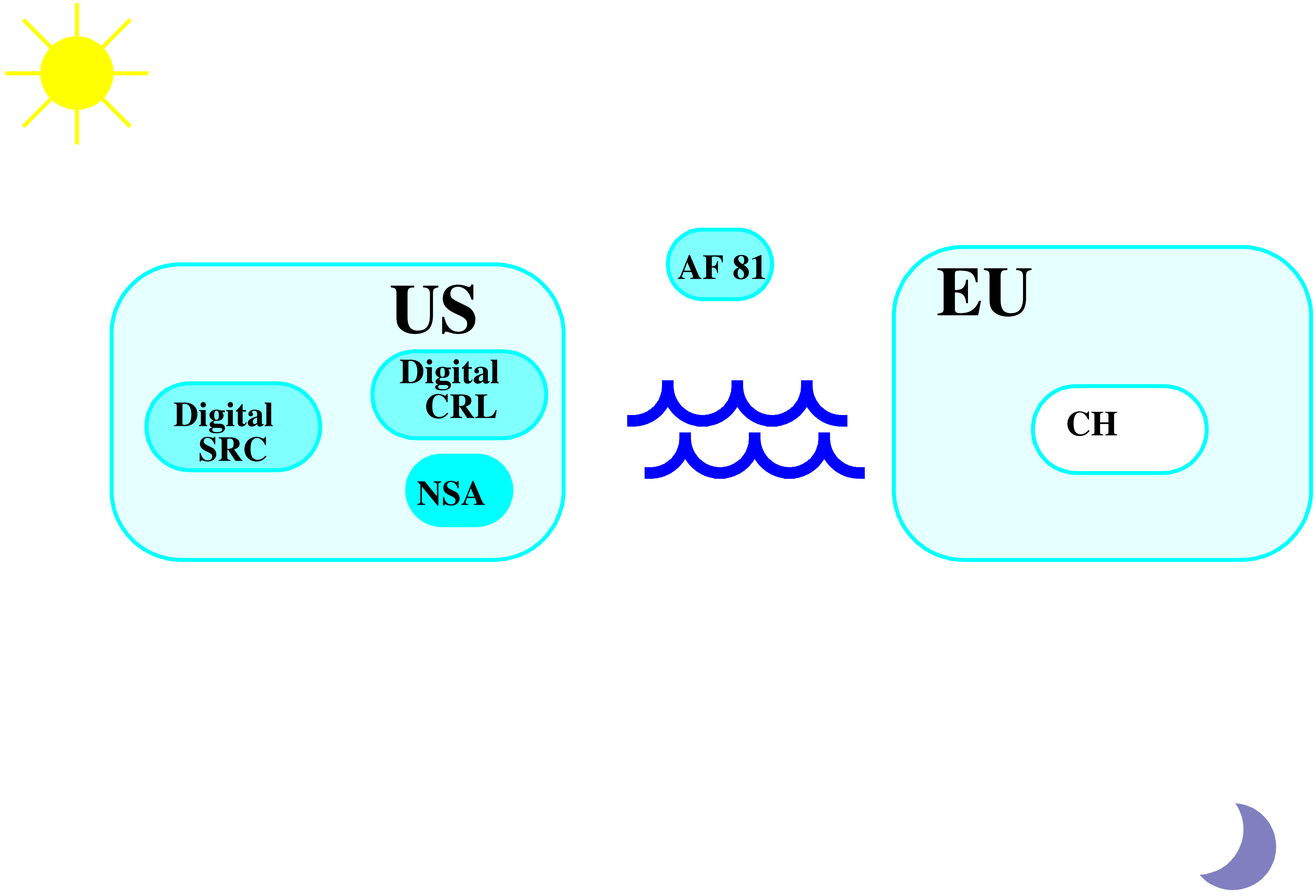
The Web



Virtual Mobility (Post-Web Software Systems)

- Basic Java Applets. (Downstream code mobility.)
- Java Servlets, and beyond. (Upstream code mobility.)
- Countless Tcl-based and Java-based ongoing projects.
- Still no (native) thread mobility. (But see, e.g., Sumatra)

The Real World (detail)



Physical Mobility (Gadgets)

- Smart cards (wired).
- Active badges, pagers (wireless).
- Cellphones (wireless).
- Palm/Laptops (wired, wireless).

Two Overlapping Views of Mobility

- Mobile Computing.
  - ~ I.e. mobile hardware, physical mobility.
- Mobile Computation.
  - ~ I.e. mobile software, virtual mobility.
- But the borders are fuzzy:
  - ~ Agents may move by traversing a network (virtually), or by being carried on a laptop (physically).
  - ~ Computers may move by lugging them around (physically), or by telecontrol software (virtually).
  - ~ Boundaries may be physical (buildings) or virtual (firewalls).

Obstacles to Mobility

- Address spaces.
  - ~ Stop pointer mobility. Circumvented by network proxies.
- Firewalls
  - ~ Stop packet mobility. Circumvented by secure tunnels.
- Sandboxes.
  - ~ Stop agent mobility. Circumvented by trust mechanisms.
- Building guards.
  - ~ Stop laptop mobility. Circumvented by removal passes.

Firewalls Everywhere

- A (nasty) fundamental change in the way we compute.
  - ~ Bye bye, flat IP addressing, transparent routing.
  - ~ Bye bye, single universal address space.
  - ~ Bye bye, transparent distributed object systems.
  - ~ Bye bye, roaming agents.
  - ~ Bye bye, action-at-a-distance computing.
- Big firewalls (for intranets), small firewalls (for applets).
- Becoming pervasive. 1 PC Firewall = $99.95.
- Firewall are designed impede access. Our task: make rightful access simple.

Summary: Mobility Postulates

- If different locations have different properties, then both people and programs 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.

Modeling Mobility

Mobility/Security Formalisms

- CSP/CCS. (Static/immutable connectivity.)

- π-calculus. (Channel mobility.)
- CHOCS. (Process mobility.)

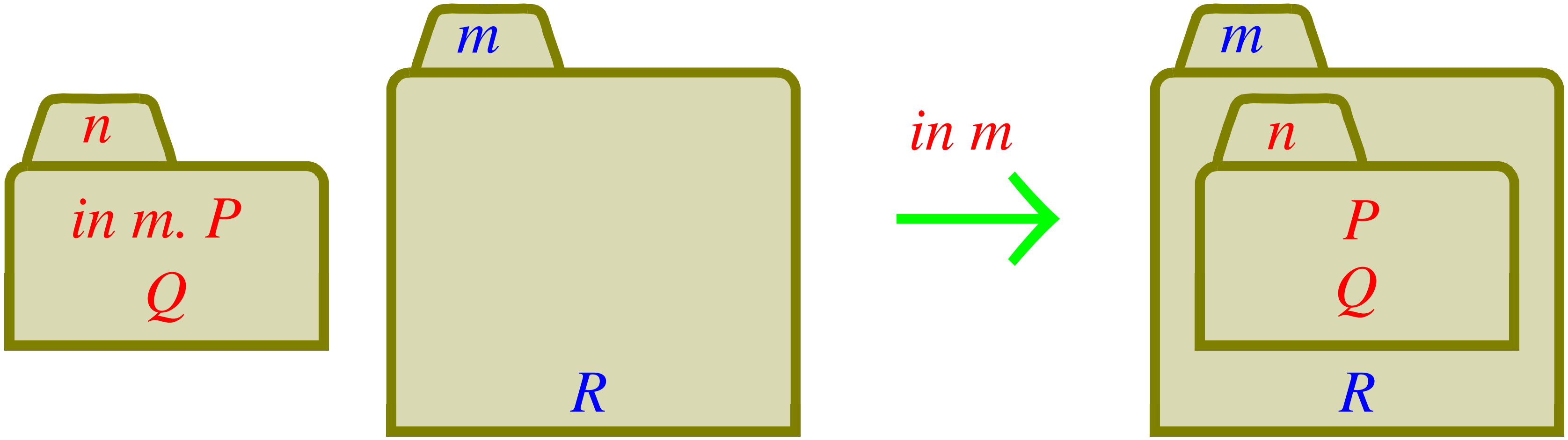
- Spi-calculus. (Channel mobility and security)
- Join calculus. (Channel mobility and locality.)
- Various calculi with failure. (Locality = Partial Failure.)

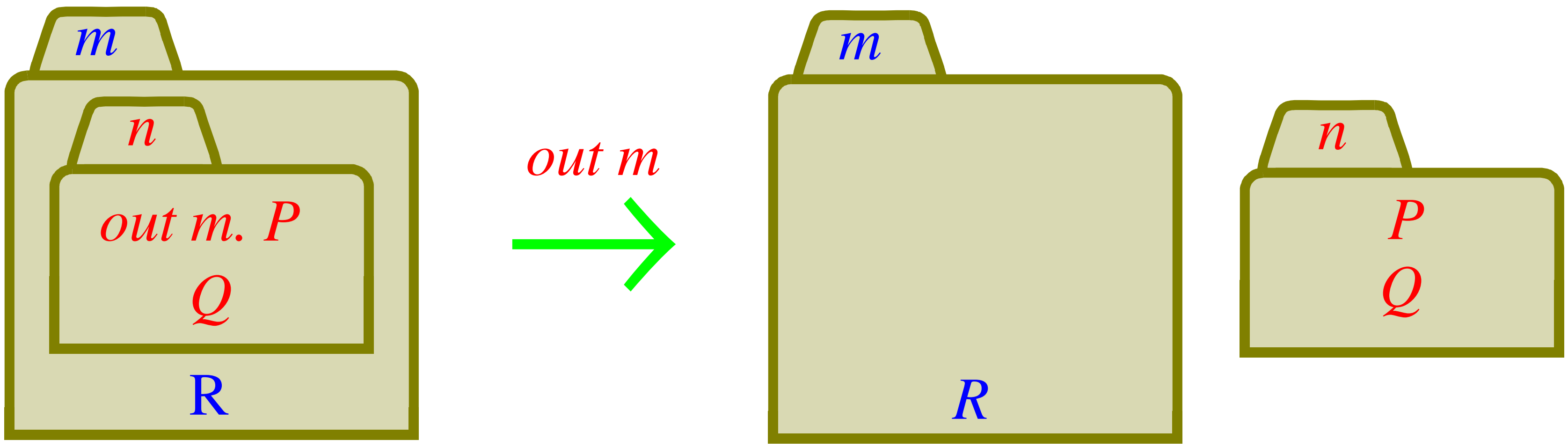
- Ambient calculus.
  (Process mobility. Locality = Topology.)

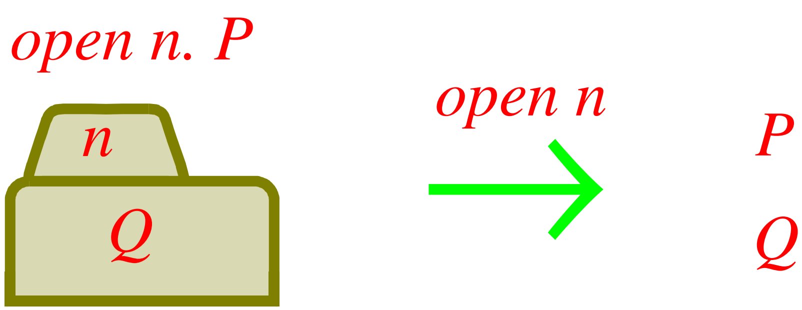
Ambients

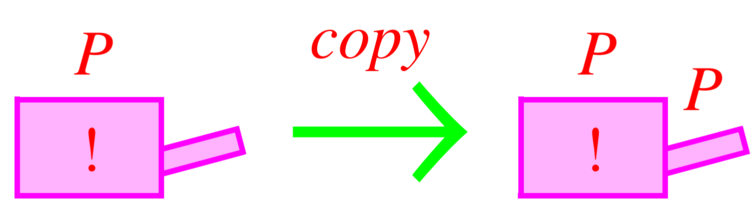
- We want to capture in an abstract way, notions of locality, of mobility, and of ability to move.
- An *ambient* is a place, delimited by a boundary, where computation happens.
- Ambients have a *name*, a collection of local *processes*, and a collection of *subambients*.
- Ambients can move in an out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

Metaphor: The Folder Calculus









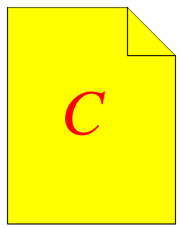
Comments

- We can look at ambients as **active folders**; each folder has a name on its tab, and can contain other folders. Each folder can also contain a whole bunch of concurrent **gremlins** that tell the folder what do and where to go. Each horizontal script line in a folder represent one (or more) gremlins.
- A folder with dynamic content can send out gremlins to find information, represented by other folders, and persuade those folders to follow the gremlins to their home folder.
- The *open* operation throws away a folder and spills its content into the current folder (where *open n.P* lives). It requires a capability *open n*, that must have been given out by folder *n*.
- The ! operation is a copy machine: if *P* is a folder, !*P* can make as many copies of *P* as desired.
- All transitions block when they cannot fire.
- The !*P* transition never blocks: it is a very idealized copy machine that never breaks and never runs out of paper. However, copying takes computation, so we can imagine that the operation is blocked until a new copy of *P* has been produced.

The set of operations on this slide (including folder creation) is Turing-complete.

Metaphor: Post-It I/O

Input: $(x).\ P\{x\}$

Output: 

$\xrightarrow{\textit{read}}$ $P\{C\}$

A Post-it can hold a *capability*:

n	a name
$in\ n$	an action capability
$out\ n$	an action capability
$open\ n$	an action capability
$C.\ C'$	a path (e.g.: $out\ n.\ in\ m$)

The Ambient Calculus

$P ::=$	an activity
$(\nu n)\ P$	new name n in a scope
0	inactivity
$P \mid P$	parallel
$!P$	replication
$a[P]$	ambient ($a ::= n$ or x)
$C.\ P$	exercise a capability
$(x).\ P$	input locally, bind to x
$\langle C \rangle$	output locally (async)

scoping

standard in process calculi

data structures

ambient-specific

actions

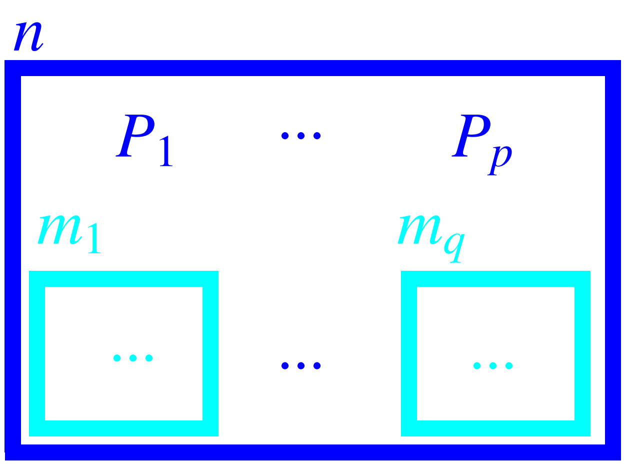
ambient I/O

$C ::=$	a capability
in a	entry capability
out a	exit capability
open a	open capability
a	name or input variable
$C.\ C'$	path

basic capabilities

useful with I/O

- Typical shape of an ambient:



n [— *name*

P_1 | ... | P_p | — *processes*

m_1[...] | ... | m_q[...] — *sub-ambients*

]

- Main operations:
  - ~ *In*. Enter an ambient. (Requires an entry capability.)
  - ~ *Out*. Exit an ambient. (Requires an exit capability.)
  - ~ *Open*. Spill the contents of an ambient. (Requires an opening capability.)

Semantics

- Behavior
  - The semantics of the ambient calculus is given in non-deterministic "chemical style" (as in Berry&Boudol's Chemical Abstract Machine, and in Milner's π-calculus).
  - The semantics is factored into a reduction relation $P \longrightarrow P'$ describing the evolution of a process P into a process P', and a process equivalence indicated by $Q \equiv Q'$.
  - Here, $\longrightarrow$ is real computation, while $\equiv$ is "rearrangement".
- Equivalence
  - On the basis of behavior, a substitutive *observational equivalence*, $P \approx Q$, is defined between processes, enabling reasoning.
  - Standard process calculi proof techniques (context lemmas, bisimulation, etc.) can be adapted.

Parallel

- Parallel execution is denoted by a binary operator:

$$P|Q$$

- It is commutative and associative:

$$P \mid Q \equiv Q \mid P$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

- It obeys the reduction rule:

$$P \longrightarrow Q \Rightarrow P \mid R \longrightarrow Q \mid R$$

Replication

- Replication is a technically convenient way of representing iteration and recursion.

$$!P$$

- It denotes the unbounded replication of a process P.

$$!P \quad \equiv \quad P \mid !P$$

- There are no reduction rules for $!P$; in particular, the process P under ! cannot begin to reduce until it is expanded out as $P|!P$.

Restriction

- The restriction operator creates a new (forever unique) ambient name n within a scope P.

$$(\nu n)P$$

- As in the π-calculus, the (νn) binder can float as necessary to extend or restrict the scope of a name. E.g.:

$$(\nu n)(P \mid Q) \;\equiv\; P \mid (\nu n)Q \quad \text{if } n \notin fn(P)$$

- Reduction rule:

$$P \longrightarrow Q \;\Rightarrow\; (\nu n)P \longrightarrow (\nu n)Q$$

Inaction

- The process that does nothing:

$$\mathbf{0}$$

- Some garbage-collection equivalences:

$$P \mid \mathbf{0} \equiv P$$

$$!\mathbf{0} \equiv \mathbf{0}$$

$$(\nu n)\mathbf{0} \equiv \mathbf{0}$$

- This process does not reduce.

Ambients

- An ambient is written as follows, where n is the name of the ambient, and P is the process running inside of it.

$$n[P]$$

- In $n[P]$, it is understood that P is actively running:

$$P \longrightarrow Q \quad \Rightarrow \quad n[P] \longrightarrow n[Q]$$

- Multiple ambients may have the same name, (e.g., replicated servers).

Actions and Capabilities

- Operations that change the hierarchical structure of ambients are sensitive. They can be interpreted as the crossing of firewalls or the decoding of ciphertexts.

- Hence these operations are restricted by *capabilities*.

$$C.\ P$$

  This executes an action regulated by the capability C, and then continues as the process P.

- The reduction rules for $C.\ P$ depend on C.

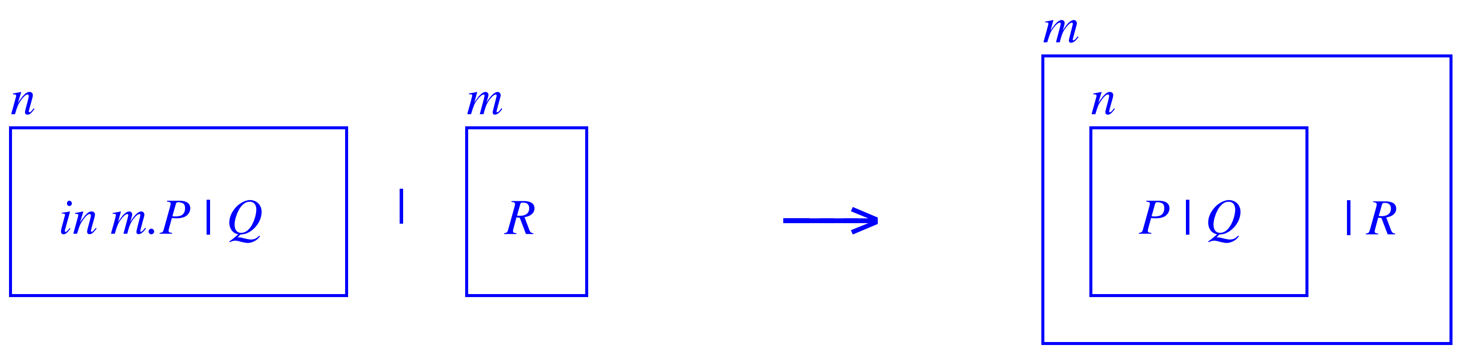
Entry Capability

- An entry capability, *in m*, can be used in the action:

$$in\ m.\ P$$

- The reduction rule (non-deterministic and blocking) is:

$$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$$



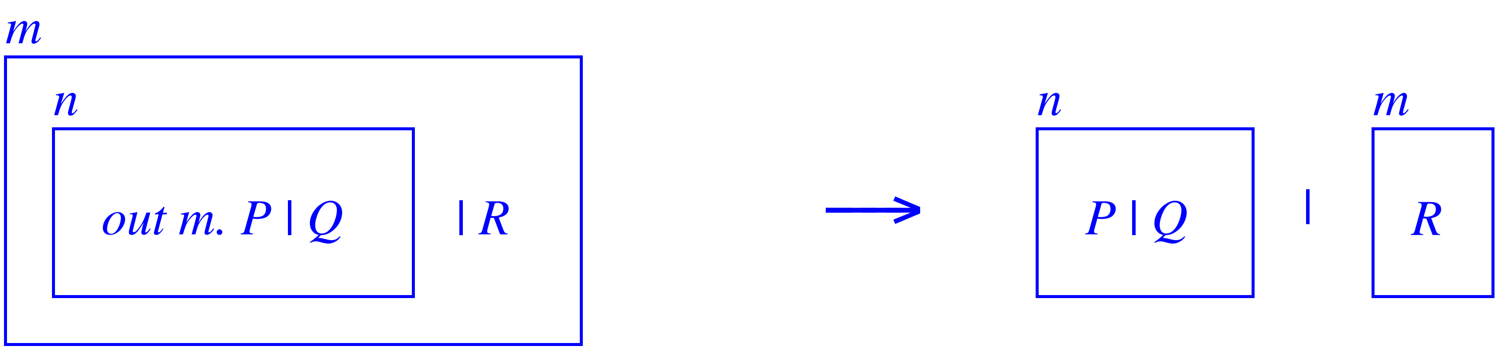
Exit Capability

- An exit capability, *out m*, can be used in the action:

$$out\ m.\ P$$

- The reduction rule (non-deterministic and blocking) is:

$$m[n[out\ m.\ P \mid Q] \mid R] \quad \longrightarrow \quad n[P \mid Q] \mid m[R]$$



Open Capability

- An opening capability, *open m*, can be used in the action:

$$open\ n.\ P$$

- The reduction rule (non-deterministic and blocking) is:

$$open\ n.\ P \mid n[Q] \;\longrightarrow\; P \mid Q$$

- An *open* operation may be upsetting to both P and Q above.
  - ~ From the point of view of P, there is no telling in general what Q might do when unleashed.
  - ~ From the point of view of Q, its environment is being ripped open.
- Still, this operation is relatively well-behaved because:
  - ~ The dissolution is initiated by the agent *open* n. P, so that the appearance of Q at the same level as P is not totally unexpected;
  - ~ *open* n is a capability that is given out by n, so $n[Q]$ cannot be dissolved if it does not wish to be.

Design Principle

- An ambient should not get killed or trapped unless:
  - It talks too much. (By making its capabilities public.)
  - It poisons itself. (By opening an untrusted intruder.)
  - It steps into quicksand. (By entering an untrusted ambient.)
- Some natural primitives violate this principle. E.g.:

$$n[\underline{burst\ n.\ P} \mid Q] \;\longrightarrow\; P \mid Q$$

Then a mere *in* capability gives a kidnapping ability:

$$entrap(C) \;\triangleq\; (\nu\, k\, m)\,(m[C.\ burst\ m.\ in\ k] \mid k[\,])$$

$$entrap(in\ n) \mid n[P] \;\longrightarrow^*\; (\nu k)\,(n[in\ k \mid P] \mid k[\,])$$

$$\longrightarrow^*\; (\nu k)\, k[n[P]]$$

Ambient I/O

- Local anonymous communication within an ambient:

$(x).\ P$	input action
$\langle C \rangle$	async output action

- We have the reduction:

$$(x).\ P \mid \langle C \rangle \longrightarrow P\{x \leftarrow C\}$$

- This mechanism fits well with the ambient intuitions.
  - ~ Long-range communication, like long-range movement, should not happen automatically because messages may have to cross firewalls and other obstacles. (C.f., Telescript.)
  - ~ Still, this is sufficient to emulate communication over named channels, etc.

Structural Equivalence Summary

$P \equiv P$	(Struct Refl)
$P \equiv Q \quad \Rightarrow \quad Q \equiv P$	(Struct Symm)
$P \equiv Q, Q \equiv R \quad \Rightarrow \quad P \equiv R$	(Struct Trans)
$P \equiv Q \quad \Rightarrow \quad (\nu n)P \equiv (\nu n)Q$	(Struct Res)
$P \equiv Q \quad \Rightarrow \quad P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \quad \Rightarrow \quad n[P] \equiv n[Q]$	(Struct Amb)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(Struct Res Res)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin fn(P)$	(Struct Res Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P] \quad \text{if } n \neq m$	(Struct Res Amb)
$P \mid \mathbf{0} \equiv P$	(Struct Zero Par)
$(\nu n)\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Res)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Repl)
$\varepsilon.P \equiv P$	(Struct ε)
$(C.C').P \equiv C.C'.P$	(Struct .)

- In addition, we identify terms up to renaming of bound names:

$$(\nu n)P \quad = \quad (\nu m)P\{n \leftarrow m\} \qquad \text{if } m \notin fn(P)$$

By this we mean that these terms are understood to be identical (for example, by choosing an appropriate representation of terms), as opposed to structurally equivalent.

Noticeable Inequivalences

- Replication creates new names:

$$!(\nu n)P \quad \not\equiv \quad (\nu n)!P$$

- Multiple n ambients have separate identity:

$$n[P] \mid n[Q] \quad \not\equiv \quad n[P \mid Q]$$

Reduction Summary

$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q$	(Red Open)
$(x).\ P \mid \langle C \rangle \longrightarrow P\{x \leftarrow C\}$	(Red Comm)
$P \longrightarrow Q \quad \Rightarrow \quad (\nu n)P \longrightarrow (\nu n)Q$	(Red Res)
$P \longrightarrow Q \quad \Rightarrow \quad n[P] \longrightarrow n[Q]$	(Red Amb)
$P \longrightarrow Q \quad \Rightarrow \quad P \mid R \longrightarrow Q \mid R$	(Red Par)
$P' \equiv P,\ P \longrightarrow Q,\ Q \equiv Q' \ \Rightarrow \ P' \longrightarrow Q'$	(Red $\equiv$)
$\longrightarrow^*$	reflexive and transitive closure of $\longrightarrow$

- An unexpected outcome.
  - The primitives invented exclusively for process mobility end up being meaningful for security. (Various caveats apply.)
  - In any case, we could extend our ambient calculus with the spi-calculus primitives, whose security features have been studied.
  - The combination of mobility and cryptography in the same formal framework seems novel and intriguing.
  - E.g., we can represent both mobility and (some) security aspects of “crossing a firewall”.

Expressiveness

- Old concepts that can be represented:
  - 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. (Being debated.)

- Net-centric concepts that can be represented:
  - Named machines and services on complex networks.
  - Encrypted data and firewalls.
  - Data packets, routing, RPC.
  - Mobile computation. (Telescript agents, applets, etc.)
  - Dynamically linked libraries.
  - Mobile devices.
  - Public transportation.

Ambients as Locks

- We can use *open* to encode locks:

$$release\ n.\ P \quad \triangleq \quad n[\,] \mid P$$

$$acquire\ n.\ P \quad \triangleq \quad open\ n.\ P$$

- This way, two processes can "shake hands" before proceeding with their execution:

$$acquire\ n.\ release\ m.\ P \mid release\ n.\ acquire\ m.\ Q$$

Ambients as Mobile Processes

$tourist \triangleq (x).\ joe[x.\ enjoy]$

$ticket\text{-}desk \triangleq\ !\langle in\ AF81SFO.\ out\ AF81CDG\rangle$

$SFO[ticket\text{-}desk \mid tourist \mid AF81SFO[route]]$

$\longrightarrow^* SFO[ticket\text{-}desk \mid$
$\quad joe[in\ AF81SFO.\ out\ AF81CDG.\ enjoy] \mid$
$\quad AF81SFO[route]]$

$\longrightarrow^* SFO[ticket\text{-}desk \mid$
$\quad AF81SFO[route \mid joe[out\ AF81CDG.\ Enjoy]]]$

Ambients as Firewalls

- Assume that the shared key k is already known to the firewall and the client.

$$Wally \triangleq (\nu\, w\, r)\, (\langle in\ r\rangle \mid r[open\ k.\ in\ w] \mid w[open\ r.\ P])$$

$$Cleo \triangleq (x).\ k[x.\ C]$$

$Cleo \mid Wally$

$$\longrightarrow^* (\nu\, w\, r)\, (\ (x).\ k[x.\ C] \mid \langle in\ r\rangle \mid r[open\ k.\ in\ w] \mid w[open\ r.\ P]\ )$$

$$\longrightarrow^* (\nu\, w\, r)\, (\ k[in\ r.\ C] \mid r[open\ k.\ in\ w] \mid w[open\ r.\ P]\ )$$

$$\longrightarrow^* (\nu\, w\, r)\, (\ r[k[C] \mid open\ k.\ in\ w] \mid w[open\ r.\ P]\ )$$

$$\longrightarrow^* (\nu\, w\, r)\, (\ r[C \mid in\ w] \mid w[open\ r.\ P]\ )$$

$$\longrightarrow^* (\nu\, w\, r)\, (\ w[r[C] \mid open\ r.\ P]\ )$$

$$\longrightarrow^* (\nu\, w)\quad (\ w[C \mid P]\ )$$

Comments

- Two secret names are introduced: *w* is the name of the firewall, and *r* is the name of a private room used as a customs checkpoint.
- We want to verify that Cleo knows the key *k*: this is done by *open k*. After that, we want to give Cleo a capability *in w* to enter the firewall. The communication of this capability must happen in a private place: we don't want some other process to snatch *in w* in transit. The private room *r* is used for this purpose.
- The room *r* has a secret name, and a single capability *in r* is made available for entering the room. Therefore we are sure that only one process enters *r* (we assume that Cleo is honest).

Turing Machine

$end[extendLft \mid S_0 \mid$

$\quad square[S_1 \mid$

$\quad\quad square[S_2 \mid$

$\quad\quad\quad ...$

$\quad\quad\quad\quad square[S_i \mid head \mid$

$\quad\quad\quad\quad\quad ...$

$\quad\quad\quad\quad\quad\quad square[S_{n\text{-}1} \mid$

$\quad\quad\quad\quad\quad\quad\quad square[S_n \mid extendRht]] \,..\,] \,..\,]]]$

The Asynchronous π-calculus

- A named channel is represented by an ambient.
  - The name of the channel is the name of the ambient.
  - Communication on a channel is becomes local I/O inside a channel-ambient.
  - A conventional name, *io*, is used to transport I/O requests into the channel.

$$(ch\ n)P \triangleq (\nu n)\ (n[!open\ io] \mid P)$$

$$n(x).P \triangleq (\nu p)\ (io[in\ n.\ (x).\ p[out\ n.\ P]] \mid open\ p)$$

$$n\langle C\rangle \triangleq io[in\ n.\ \langle C\rangle]$$

- These definitions satisfy the expected reduction:

$$n(x).P \mid n\langle C\rangle \longrightarrow^* P\{x{\leftarrow}C\}$$

in presence of a channel for *n*.

- Therefore:

$$\langle\!\langle (\nu n)P \rangle\!\rangle \triangleq (\nu n)\,(n[!open\ io] | \langle\!\langle P \rangle\!\rangle)$$

$$\langle\!\langle n(x).P \rangle\!\rangle \triangleq (\nu p)\,(io[in\ n.\,(x).\,p[out\ n.\,\langle\!\langle P \rangle\!\rangle]] | open\ p)$$

$$\langle\!\langle n\langle m\rangle \rangle\!\rangle \triangleq io[in\ n.\,\langle m\rangle]$$

$$\langle\!\langle P|Q \rangle\!\rangle \triangleq \langle\!\langle P \rangle\!\rangle | \langle\!\langle Q \rangle\!\rangle$$

$$\langle\!\langle !\mathrm{P} \rangle\!\rangle \triangleq !\langle\!\langle \mathrm{P} \rangle\!\rangle$$

~ The choice-free synchronous π-calculus, can be encoded within the asynchronous π-calculus.

~ The λ-calculus can be encoded within the asynchronous π-calculus.

Contextual Equivalence

- Exhibition

$$P{\downarrow}n \quad \Leftrightarrow \quad P \equiv (\nu n_1 \ldots n_p)(n[Q] \mid R) \;\wedge\; n \notin \{n_1 \ldots n_p\}$$

- Convergence

$$P{\Downarrow} \quad \Leftrightarrow \quad \exists n.\; P \longrightarrow^{*} Q \;\wedge\; Q{\downarrow}n$$

- Contextual Equivalence

$$P \approx Q \quad \Leftrightarrow \quad \forall \boldsymbol{C}\{\!\!\{\bullet\}\!\!\}.\; \boldsymbol{C}\{\!\!\{P\}\!\!\}{\Downarrow} \Leftrightarrow \boldsymbol{C}\{\!\!\{Q\}\!\!\}{\Downarrow}$$

Security Applications

Firewalls

- $n[P]$ is a firewall named n protecting P.
- $in\ n$ is the capability needed to enter the firewall.
- $out\ n$ is the capability needed to exit the firewall.
- The *context* is the Internet.
- The Perfect-Firewall Equation:

$$(\nu n)\, n[P] \approx \mathbf{0} \qquad \text{(if } n \text{ not in } P\text{)}$$

Cryptography

- The ambient calculus can, without special extensions, model certain cryptographic procedures.
  - ~ In particular, it can model the most basic subset of the spi-calculus:

{M}N	shared-key encryption of M by N
decrypt M with N	shared-key decryption

- It does not embrace a particular implementation:
  - ~ It does not model the ability an attacker may have to compare bit patterns.
  - ~ It does not model the ability an attacker may have to exploit properties of a specific underlying crypto.

Nonces

- A nonce is simply a fresh name that can, for example, be communicated by an output action.

  $Q \mid (\nu n)\ (\langle n \rangle \mid P)$ output a nonce n for Q

  When the nonce comes back to P, it can be verified by *open* n.

Shared Keys

- A name can be used as a shared key, as long as it is kept secret and shared only by certain parties.

$k[\langle txt \rangle]$	encrypt *txt* with the shared key *k*
open k. (x). *P*	decrypt with the shared key *k* and read the message

- Anybody who knows *k* can decrypt a message $k[\langle txt \rangle]$:
  - Either by *open k* (destructively).
  - Or by *in k* followed by *out k* (non-destructively).

Public Keys: Signed Messages

- If $k[\langle txt \rangle]$ is the plaintext *txt* encrypted by k, then *open* k represents the (public) ability to open a k-envelope, without knowing k.

Principal A	
(νk)	create a new signature key
$!\langle open\ k \rangle$	publish the signature verifier
$\mid k[\langle txt \rangle]$	sign a message
Principal B	
$(open\text{-}cap).$	acquire the signature verifier
$open\text{-}cap.$	verify an available message
$(msg).\ P$	read the message and proceed

Public Keys: Coded Message

- If $k[\langle txt \rangle]$ is the plaintext *txt* encrypted by k, then $(x).\ k[\langle x \rangle]$ represents the (public) ability to insert a plaintext in a k-envelope, without knowing k.

Principal A	
(νk)	create a new encryption key
$!(x).\ k[\langle x \rangle]$	publish message encryptors (possibly route them)
$\mid\ !open\ k.\ (x).\ P$	decrypt incoming messages and proceed
Principal B	
$\langle txt \rangle$	encrypt a message for A
	(assuming an encryptor for A is available here)
	(possibly route it back to A)

Ciphers

- $k[\langle txt \rangle]$ is the plaintext *txt* encrypted with key k.
- $P \approx Q$ means "no attacker can tell P from Q".
- The Perfect-Cipher Equation:

$$(\nu k_1)\ k_1[\langle txt_1 \rangle] \ \approx\ (\nu k_2)\ k_2[\langle txt_2 \rangle]$$

  - Simply because $(\nu k_1)\ k_1[\langle txt_1 \rangle] \approx \mathbf{0} \approx (\nu k_2)\ k_2[\langle txt_2 \rangle]$.
  - This is a consequence of (a) the reductions allowed in the calculus, (b) the absence of other reductions that might make distinctions, (c) the (debatable) interpretation of ambient operations as crypto operations.

Calculi vs. Reality

- Calculi make "implicit security assumptions".
  - ~ *Nominal calculi*, like π, spi, assume that <u>nobody</u> can guess the name of a private channel.
  - ~ The ambient calculus assumes that <u>nobody</u> can extract a name from a capability.
  - ~ Consequences include the perfect-cipher equation.
- A) This is good.
  - ~ These assumptions are "security abstraction" that enable high-level reasoning (via ≈).
  - ~ These assumptions can be realized by different implementation (crypto) techniques.
  - ~ They may increase practical security by providing a programming model that is more transparent.

- B) This is bad.
  - Such assumption are dangerous since they are not obviously "realistic". How do they map to algebraic properties of the underlying crypto primitives?
  - They may hold within the calculus, but do they keep holding under low-level attacks (if somebody can dissect an agent)?
- (Speculation.) Implicit security assumptions must be made explicit and must be "securely implemented".
  - One must describe an implementation of the calculus in terms of realistic cryptographic primitives.
  - One must prove that the implementation is (1) correct and (2) prevents certain low-level attacks. [Abadi, Gonthier, Fournet]

Language Applications

Ambient-like Languages

- No “hard” pointers.

  All references are URLs, symbolic links, or such.

- Migration/Transportation

  Thread migration.

  Data migration.

  Whole-application migration.

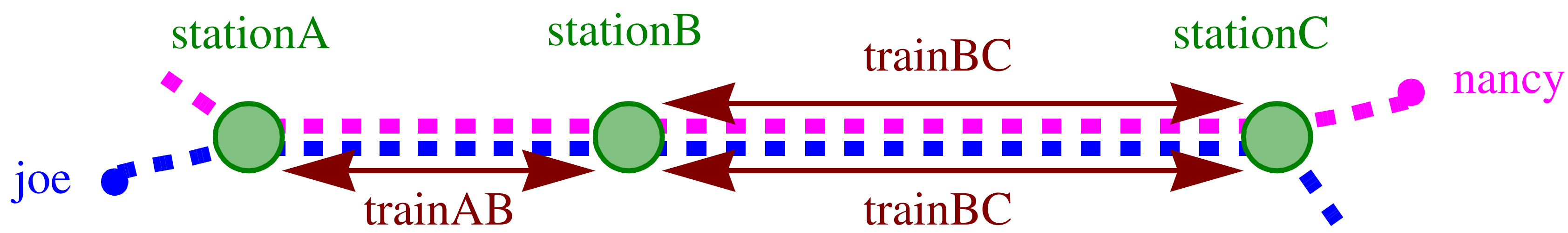
- Dynamic linking.

  A missing library or plug-in may suddenly show up.

- No communication exceptions.

  Blocking/exactly-once semantics.

Transportation




```
let train(stationX stationY XYatX XYatY tripTime) =
  new moving.       // assumes the train originates inside stationX
    moving[rec T.
      be XYatX. wait 2.0.
      be moving. go out stationX. wait tripTime. go in stationY.
      be XYatY. wait 2.0.
      be moving. go out stationY. wait tripTime. go in stationX.
      T];

new stationA stationB stationC ABatA ABatB BCatB BCatC.
  stationA[ train(stationA stationB ABatA ABatB 10.0) ] |
  stationB[ train(stationB stationC BCatB BCatC 20.0) ] |
  stationC[ train(stationC stationB BCatC BCatB 30.0) ] |
```

```
new joe.
  joe[
    go in stationA.
    go in ABatA. go out ABatB.
    go in BCatB. go out BCatC.
    go out stationC] |

new nancy.
  nancy[
    go in stationC.
    go in BCatC. go out BCatB.
    go in ABatB. go out ABatA.
    go out stationA]
```

Execution trace

```
moving: Be ABatA
moving: Be BCatC
moving: Be BCatB
nancy: Moved in stationC
nancy: Moved in BCatC
joe: Moved in stationA
joe: Moved in ABatA
ABatA: Be moving
BCatC: Be moving
moving: Moved out stationC
BCatB: Be moving
moving: Moved out stationB
moving: Moved out stationA
moving: Moved in stationB
moving: Be ABatB
joe: Moved out ABatB
ABatB: Be moving
moving: Moved out stationB
moving: Moved in stationC
moving: Be BCatC
BCatC: Be moving
moving: Moved out stationC
moving: Moved in stationA
moving: Be ABatA
ABatA: Be moving
moving: Moved out stationA
moving: Moved in stationB
```

```
moving: Be BCatB
nancy: Moved out BCatB
joe: Moved in BCatB
BCatB: Be moving
moving: Moved out stationB
moving: Moved in stationB
moving: Be ABatB
nancy: Moved in ABatB
ABatB: Be moving
moving: Moved out stationB
moving: Moved in stationB
moving: Be BCatB
BCatB: Be moving
moving: Moved out stationB
moving: Moved in stationA
moving: Be ABatA
nancy: Moved out ABatA
nancy: Moved out stationA
ABatA: Be moving
moving: Moved out stationA
moving: Moved in stationB
moving: Be ABatB
moving: Moved in stationC
moving: Be BCatC
joe: Moved out BCatC
joe: Moved out stationC
moving: Moved in stationC
...
```

Conclusions

- The notion of *named, active, hierarchical, mobile ambients* captures the structure of complex networks and of mobile computing/computation.
- The ambient calculus formalizes ambient notions simply and powerfully.
  - It is no more complex than common process calculi.
  - It supports reasoning about mobility and (hopefully) security.
- We can now envision new programming methodologies/ libraries/languages for global computation.

Extra

Locality and Computation

- “Model of Computation” = “What can be observed”.
- Observable Locality.
  - Survival.
  - Resource availability.
  - Meaning of names/references.
  - Barriers.
- Observable Quality of Service (Between Locations).
  - Latency and bandwidth.
  - Price.
  - Convenience.

- Examples
  - ~ An active badge enters a room.
  - ~ A infrared-enabled PDA enters a room.
  - ~ A wireless laptop enters a building.
  - ~ A mobile phone enters a cell.
  - ~ A smart card enters an NC.
  - ~ A Java applet enters a firewall, then a browser.

Movement from the Inside or the Outside: Subjective vs. Objective

There are two natural kinds of movement primitives for ambients. The distinction is between "I make you move" from the outside (*objective move*) or "I move" from the inside (*subjective move*). Subjective moves, the ones we have already seen, obey the rules:

$$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$$
$$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$$

Objective moves (indicated by an *mv* prefix), instead obey the rules:

$$mv\ in\ m.\ P \mid m[R] \longrightarrow m[P \mid R]$$
$$m[mv\ out\ m.\ P \mid R] \longrightarrow P \mid m[R]$$

These two kinds of move operations are not trivially interdefinable. The objective moves have simpler rules. However, they operate only on ambients that are not active; they provide no way of moving an existing running ambient. The subjective moves, in contrast, cause active ambients to move and, together with *open*, can approximate the effect of objective moves (as we discuss later).

Another kind of objective moves one could consider is:

$$mv\ n\ in\ m.\ P \mid n[Q] \mid m[R] \longrightarrow P \mid m[n[Q] \mid R]$$
$$m[mv\ n\ out\ m.\ P \mid n[Q] \mid R] \longrightarrow P \mid m[P \mid R] \mid n[Q]$$

These are objective moves that work on active ambients. However they are not as simple as the previous objective moves and, again, they can be approximated by subjective moves and *open*.

In examining these variations, one should consider who has the authority to move whom. In the case of the subjective moves, the authority rests in the top-level agents of an ambient, which naturally act as *control agents* for the ambient. In the case of objective moves, one should be careful to require enough capabilities so that ambients cannot be arbitrarily kidnapped.

Ambients as Storage Cells

A cell *cell c v* stores a value *v* at a location *c*, where a value is a capability. The cell is set to output its current contents destructively, and is set to be "refreshed" with either the old contents (by *get*) or a new contents (by *set*).

$$\begin{aligned} \mathit{cell}\ c\ v &\triangleq c^{\downarrow\uparrow}[\langle v\rangle \mid !(x).\ \langle x\rangle] \\ \mathit{get}\ c\ (x).\ P &\triangleq \mathit{mv\ in}\ c.\ (x).\ (\langle x\rangle \mid \mathit{mv\ out}\ c.\ P) \\ \mathit{set}\ c\ \langle v\rangle.\ P &\triangleq \mathit{mv\ in}\ c.\ (x).\ (\langle v\rangle \mid \mathit{mv\ out}\ c.\ P) \end{aligned}$$

Note that *set* is essentially an output operation, but it is a synchronous one: its sequel *P* runs only after the cell has been set.

Parallel *get* and *set* operations do not interfere. It is also possible to code an atomic *get-and-set* primitive, which could be used to code *test-and-set*; in that case the value expression *v* below would contain a test depending on *x*.

$$\mathit{get\text{-}and\text{-}set}\ c\ (x)\ \langle v\rangle.\ P \triangleq \mathit{mv\ in}\ c.\ (x).\ (\langle v\rangle \mid \mathit{mv\ out}\ c.\ P)$$

0.0.1 Records

A record is a named collection of cells. Since each cell has its own name, those names can be used as field labels:

$$record\ r(l_1{=}v_1\ \ldots\ l_n{=}v_n)\ \triangleq\ r^{\downharpoonright\upharpoonright}[cell\ l_1\ v_1 \mid \ldots \mid cell\ l_n\ v_n]$$

$$getr\ r\ l\ (x).\ P\ \triangleq\ mv\ in\ r.\ get\ l\ (x).\ mv\ out\ r.\ P$$

$$setr\ r\ l\ \langle v\rangle.\ P\ \triangleq\ mv\ in\ r.\ set\ l\ \langle v\rangle.\ mv\ out\ r.\ P$$

A record can contain the name of another record in one of its fields. Therefore sharing and cycles are possible.