stiamo cercando di comprendere un modello di programmazione e poi conoscere le primitive linguistiche necessarie per programmare servizi distribuiti FaaS, vogliamo avere delle politiche di sicurezza che definiscono localmente delle proprietà di sicurezza che definiscono l'accesso a risorse critiche e vogliamo avere un meccanismo di tali servizi senza stato in cui l'invocazione viene fatta non conoscendo il nome del servizio ma le proprietà che il servizio deve soddisfare. In più viviamo in un contesto in cui vorremmo definire gli orchestratori che danno il tick, cioè dice a tutti gli elementi cosa fare per far funzionare l'applicazione. Il meccanismo di orchestrazione aiuta nella operazione di richiesta per contratto quindi deve certificare i contratti dei servizi e dall'altra parte deve fare il piano dell'orchestrazione ovvero fare il mapping della descrizione del programma in una qualche struttura sottostante.

SUMMING UP…

a programming model for secure service composition:

- **distributed** stateless services
- **safety framings** scoped policies on localized execution histories
- **req-by-contract** service invocation

static orchestrator:

- certifies the **behavioural interfaces** of services
- provides a client with the **viable plans** driving secure executions

WHAT'S NEXT

- programming: syntax and **operational semantics**
- static semantics: **type & effect system**
  - **types** carry annotations H about service behaviour
  - **effects** H are history expressions, which over-approximate the actual execution histories
  - **Combination of EM and Static Analysis**
- extracting viable plans:
  - **linearization:** unscrambling the structure of H
  - **model checking:** valid plans are viable

SERVICES: FUNCTIONAL + ….

Services	e ::= x	variable
	α	access event
	if b then e else e′	conditional
	λ x.e	Lambda
	λ_z x.e	Recursive Lambda
	e e′	application

da un nostro punto di vista sia i servizi che l'orchestratore sono scritti in un linguaggio che ha un parte funzionale. Questa è una sintassi astratta, possiamo dichiarare variabili, il condizionale e quello che è il numero di eventi di accesso a risorse critiche indicato con alfa, vuol dire che nel nostro linguaggio di programmazione sappiamo cosa vuol dire accedere a un DB, aprire un file, oppure sappiamo cosa vuol dire aprire una connesisone di rete. Abbiamo dei marcatori che ci informano che stiamo effettuando delle azioni sulle risorse critiche. Questa è una operazione che astrae dalle risorse. I lambda sono funzioni non ricorsive anonime, tipica caratteristica dei linguaggi funzionali, lambda_z è una funzione ricorsiva in cui diciamo anche il nome della funzione. Poi abbiamo l'applicazione di funzione e, che corrisponde alla applicazione funzionale, con parametro formale e' dove l'espressione e e' si può vedere come applicazione di funzione perchè le espressioni vengono valutate. Il parametro viene valutato quando ottiene un valore.

NOTE: LAMBDA

ANONYMOUS FUNCTIONS λ **x.e**

λ **x. x+1**

OCAML NOTATION

```
fun x = e
fun x = x + 1
```

NOTE: LAMBDA

RECURSIVE FUNCTIONS λ_z x.e

OCAML NOTATION

```
let rec fact x =
        if x <= 1 then 1 else x * fact (x - 1);;
```

NOTATION

λ .e stands for for λx.e, for x not in fv(e).

The following abbreviation is standard:
e1;e2′ = (λ .e2′)e1.

Useremo spesso questa notazione: lambda spazio e, per un lambda dove ho una variabile x che però non compare nel corpo voglio definire cioè una funzione senza parametri. Dovendo farlo sintenticamente, mi ricordo che quel parametro non comparirà nel corpo. e1;e2 è uguale alla funzione senza parametri che ha come corpo e2 a cui applico e1. Applico il parametro e1 alla funzione lambda e2 quindi opero per call by value, il valore della valutazione di e1 viene passata alla funzione quindi vuol dire che viene eseguito il corpo della funzione che è e2 senza fare il passaggio dei parametri, quindi sto eseguendo prima e1 e poi e2 quindi il tipo che ottengo come risultato sostanzialmente è il tipo di e2 correttamente rispetto alla applicazione funzionale.

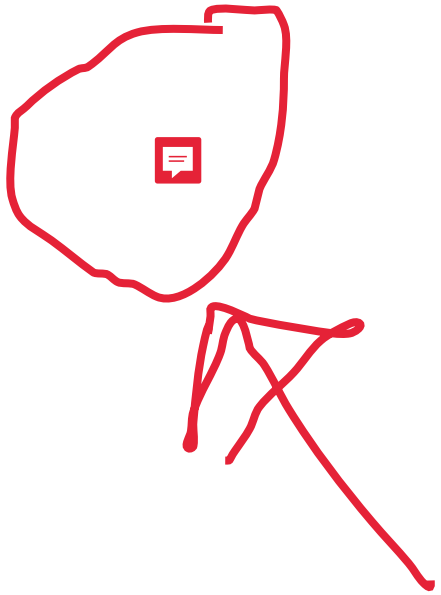
SERVICES: FUNCTIONAL + SECURITY

Services	e ::= x	variable
	α	access event
	if b then e else e′	conditional
	λ x.e	lambda
	λ_z x.e	Recursive lambda
	e e′	application
	φ[e]	safety framing

SERVICES: FUNCTIONAL + SECURITY + COMS

Services	$e ::= x$	variable
	α	access event
	if b then e else e′	conditional
	$\lambda\ x.e$	Lambda
	$\lambda_z\ x.e$	Recursive lambda
	$e\ e'$	application
	$\varphi[e]$	safety framing
	$req_r\ \tau$	service request
(only in configs)	wait ℓ	wait reply

NETWORK SOFTWARE ARCHITECTURE



location

service code and published interface

$$N ::= \ell\{e{:}\tau\}{:}\ \eta,\ e'$$

$$N \parallel N'$$

published service

composition

running code

execution history

PLANS: MAPPING REQUESTS TO SERVICE ENDPOINTS

A **plan** is a function from requests **r** to services ℓ

Noi abbiamo detto che chi programma lo fa a un livello astratto ovvero non vede l'architettura sottostante. Sarà poi compito del orchestratore fare il mapping tra il programma del cliente che usa informazioni simboliche e gli endpoint che sono poi la rappresentazione puntuale all'interno delle infrastrutture di networking di dove stanno i servizi. Allora, per fare questo, il runtimne deve avere un meccanismo per descrivere il placement. Vuol dire che nella trusted computer base ci deve essere anche il placement che definiamo sinteticamente in questo modo. Lo chiamiamo plan e registra le associazioni tra le etichette delle richieste che compaiono nel codice del cliente, quello che abbiamo visto quando abbiamo introdotto la nozione di richiesta che ha un'etichetta che mi dice il punto della richiesta e poi il tipo della richiesta (slide 8) quindi un piano che cosa fa? associa alla particolare richiesta r esattamente l'endpoint quindi l'indirizzo logico l ancora una volta di dove e' il servizio. Quindi vuol dire che possiamo mettere insieme varie nozioni di binding nome della richiesta-servizio. Tecnicamente c'è una piccola nota tecnica che qui è nascosta in realtà quando uno definisce il piano ha già un'idea di qual'è l'informazione di networking a cui fa riferimento nel senso non può fare un'applicazione sull'universo mondo di internet sull'universo mondo del cloud quindi di fatto ha un sottoinsieme di locazioni che sta considerando. Questo è rappresentato da un ordine parziale che assumiamo tra le locazioni dei servizi. Quindi è un modo per dire che è vero che si fa un'applicazione distribuita, ma non sull' universo mondo. Abbiamo un po' di informazione su dove stanno i servizi.

π	::=	**0**	empty
		r$[\ell]$	service choice
		$\pi \mid \pi'$	composition

Plans respect the partial knowledge $\ell < \ell'$ of services about the network ($<$ is a partial ordering)

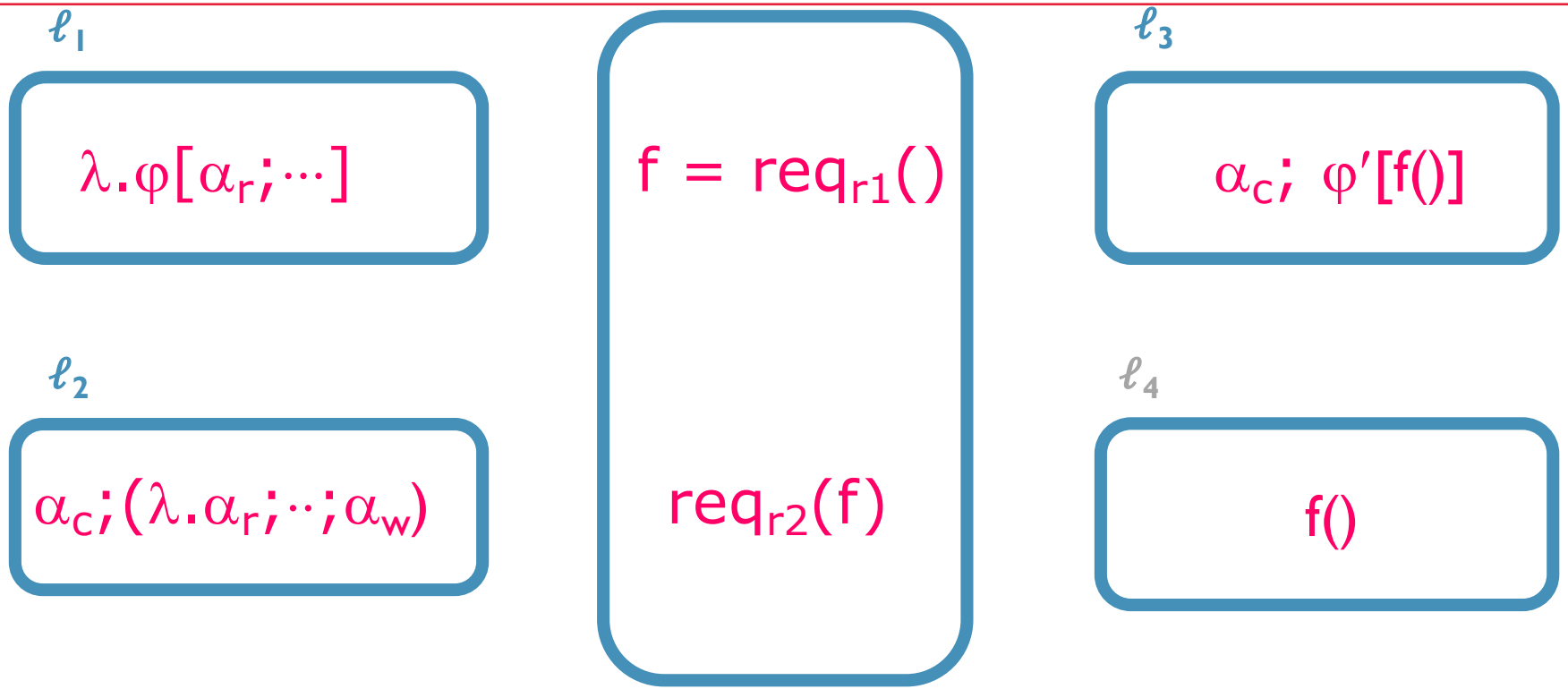
EXAMPLE

- The client application
  - Search for a suitable code
  - Delegate its execution

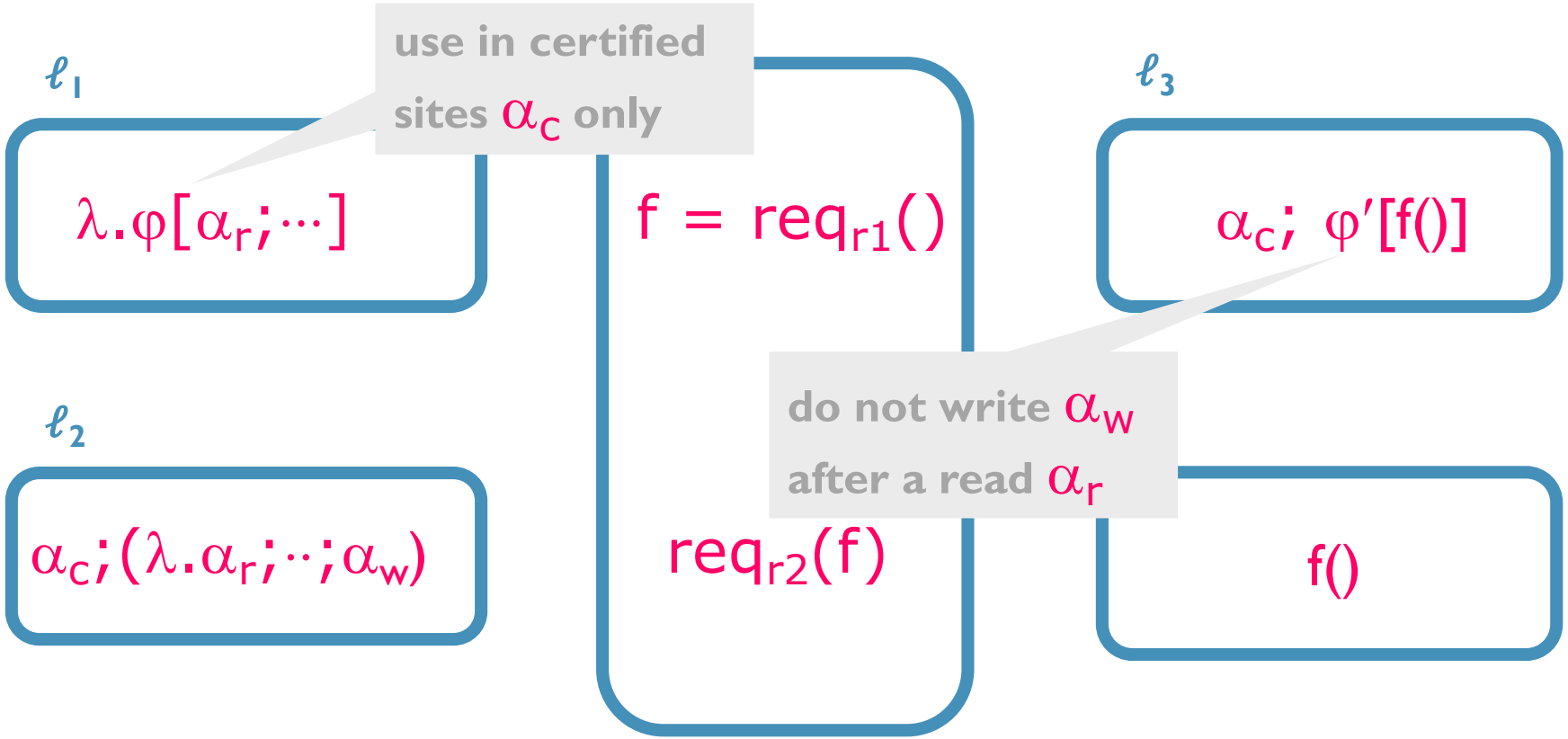
Avevamo fatto dei use case la volta scorsa, quando avevamo definito i principi che ci stavano guidando a questo modello di programmazione erano la esecuzione remota. Quindi io so sono un'entita' esecuzione. Non ho potenza di calcolo sufficiente. Devo fare delle operazioni su dati, mando il codice ad un servizio che ha potenza di calcolo. L'altra possibilità era di avere del codice mobile, cioè di eseguire all'interno del proprio ambiente di esecuzione un codice che abbiamo preso da fuori. Ora facciamo vedere con queste notazioni che abbiamo introdotto un'applicazione cliente e che cosa fa? Va alla ricerca del codice che gli può servire per risolvere i suoi problemi e poi mandarla in esecuzione. Quindi esattamente tutti i due aspetti che abbiamo visto la volta scorsa come use case de sistema che vogliamo costruire in un unico punto.

EXAMPLE: DELEGATING CODE EXECUTION

Viviamo nel nostro mondo, dove abbiamo quattro servizi. Quindi vedete quel discorso che dicevo prima del fatto che abbiamo un'informazione specifica di dove stanno i servizi, non stanno sull'universo mondo intero. Alla locazione l1 abbiamo un servizio lambda caratterizzato da una propria politica di sicurezza, la PHI dice che posso usare questo codice soltanto in siti certifcati. Quindi questo è il servizio l1 senza parametri. Il servizio l2 invece è un servizio che prima si certifica, quindi esegue un'azione di certificazione. Vuol dire che mette in un qualche modo una firma che certifica delle proprieta' dei servizi. Queste qui sono le informazioni che sono presenti nei grant di Java. Qui abbiamo un pochino astratto perché non vogliamo avere il dettaglio sintattico, vogliamo solo cercare di capire l'idea. Dopo essersi certificato, abbiamo tutte funzioni senza parametri, che esegue una sequenza di operazioni critiche sulle risorse. Fa un read inizialmente, poi termina con un write. Alla locazione l3 abbiamo un servizio che anche lui si certifica e poi esegue una funzione f che in questo momento è sconosciuta, vuol dire che è una funzione che gli deve essere fornita e questa funzione senza paramenti all'interno di una politica PHI' la quale dice che non ho write after read. Infine l4 esegue semplicemente la funzione.

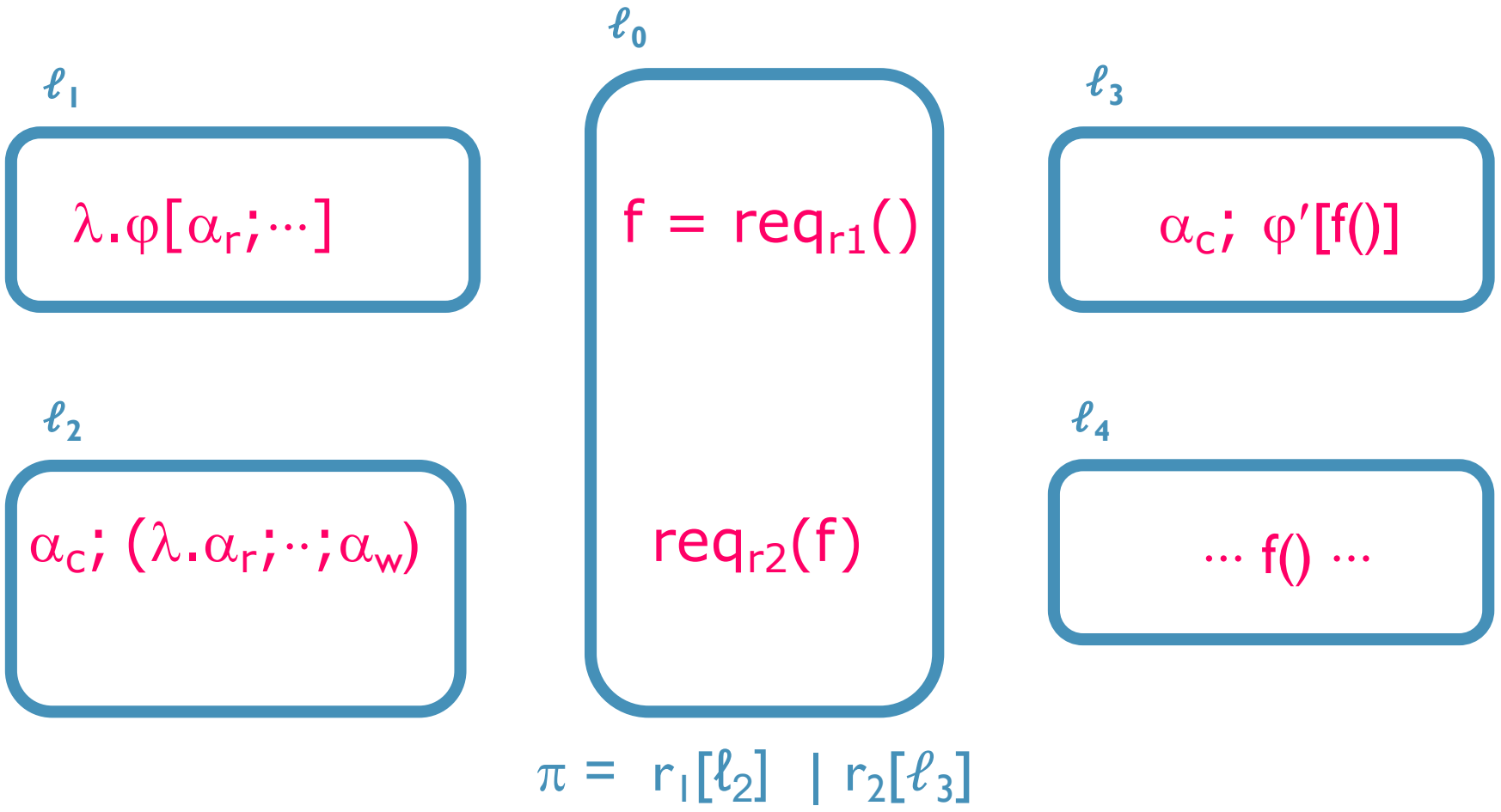


EXAMPLE: DELEGATING CODE EXECUTION

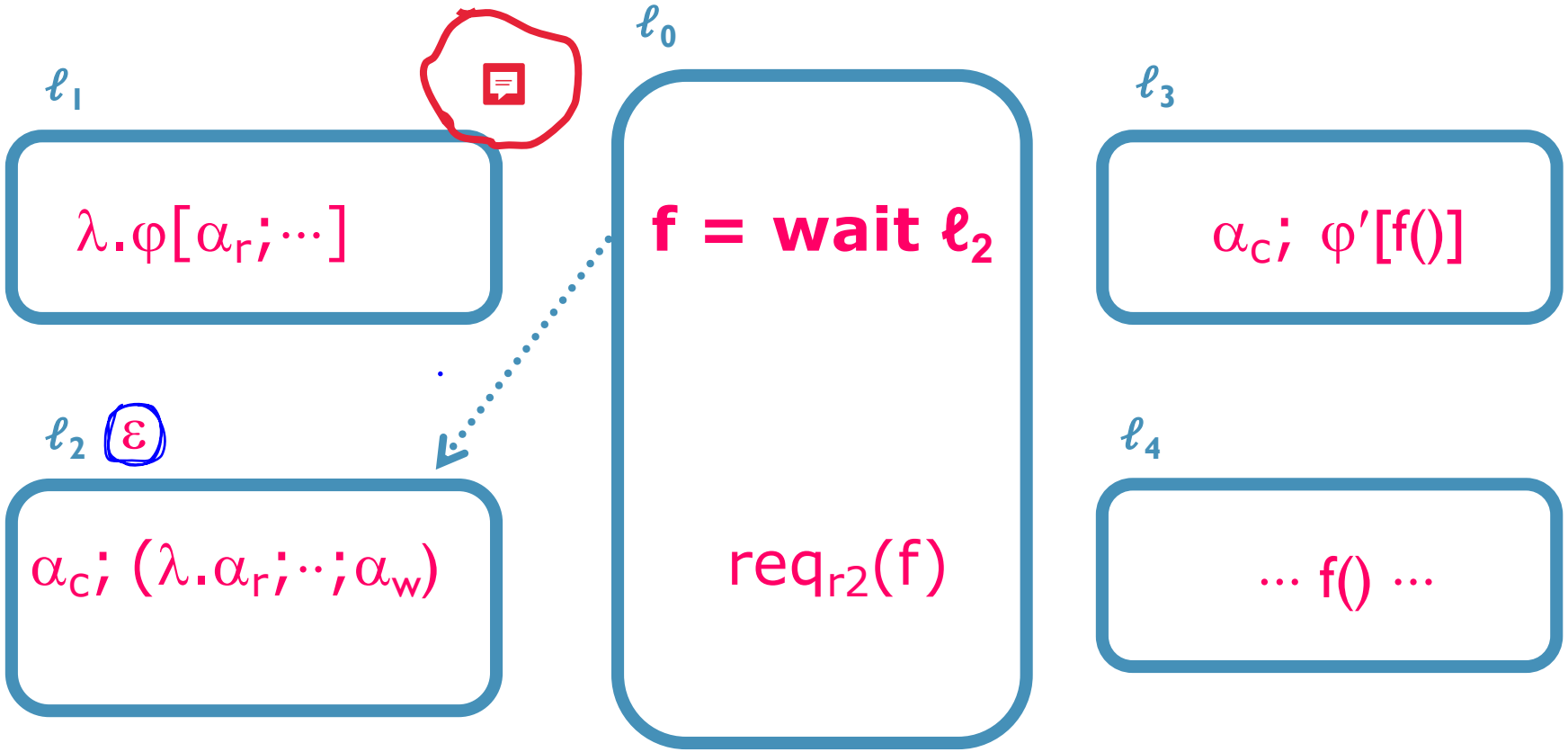


Assumiamo un mapping che mi dice: la richiesta del servizio l1 mi viene risolta a l2 e la richiesta a l2 mi viene risolta a l3. Quindi il programma di orchestrazione è astratto. Non vedo cosa ho ad l1 e cosa è associato a l2. Però quando la vado ad eseguire lo faccio tenendo una nozione di placement, quella descritta dal pipe che mi dice che l'endpoint di l1 è il servizio alla locazione l2, l'endpoint di l2 è il servizio alla locazione l3. Potete immaginare che ci sia un omino, l'operatore di controllo della rete che ha fatto il placement. In realtà c'è un operator control che è un programma che cerca di ottimizzare in base alla richiesta e al carico che c'è attualmente nel sistema. Però di fatto abbiamo un meccanismo che mi permette di fare il placement e poi andremo molto più avanti a vedere come possiamo automatizzare questo aspetto del placement.

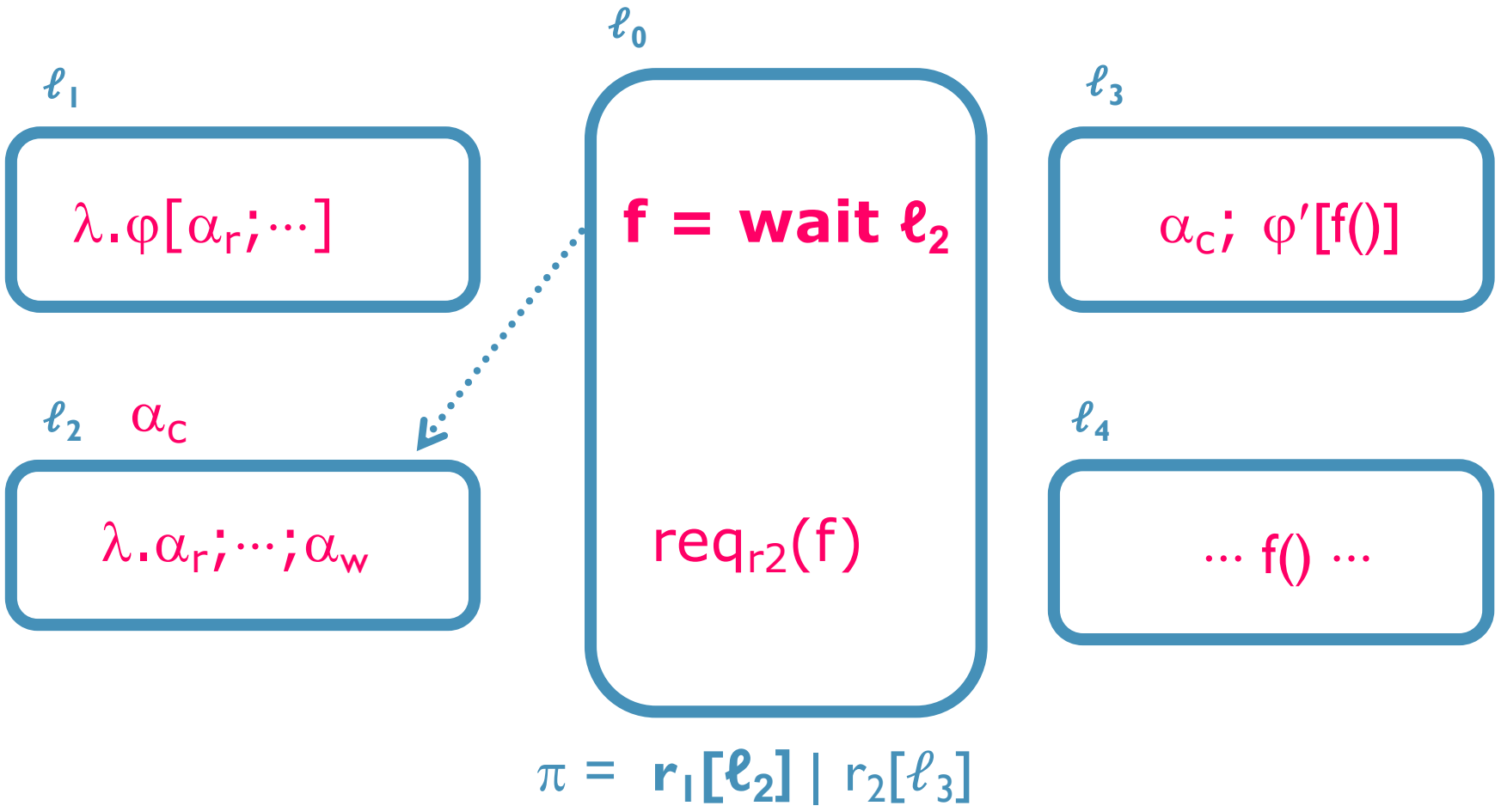
EXECUTING A NETWORK OF SERVICES



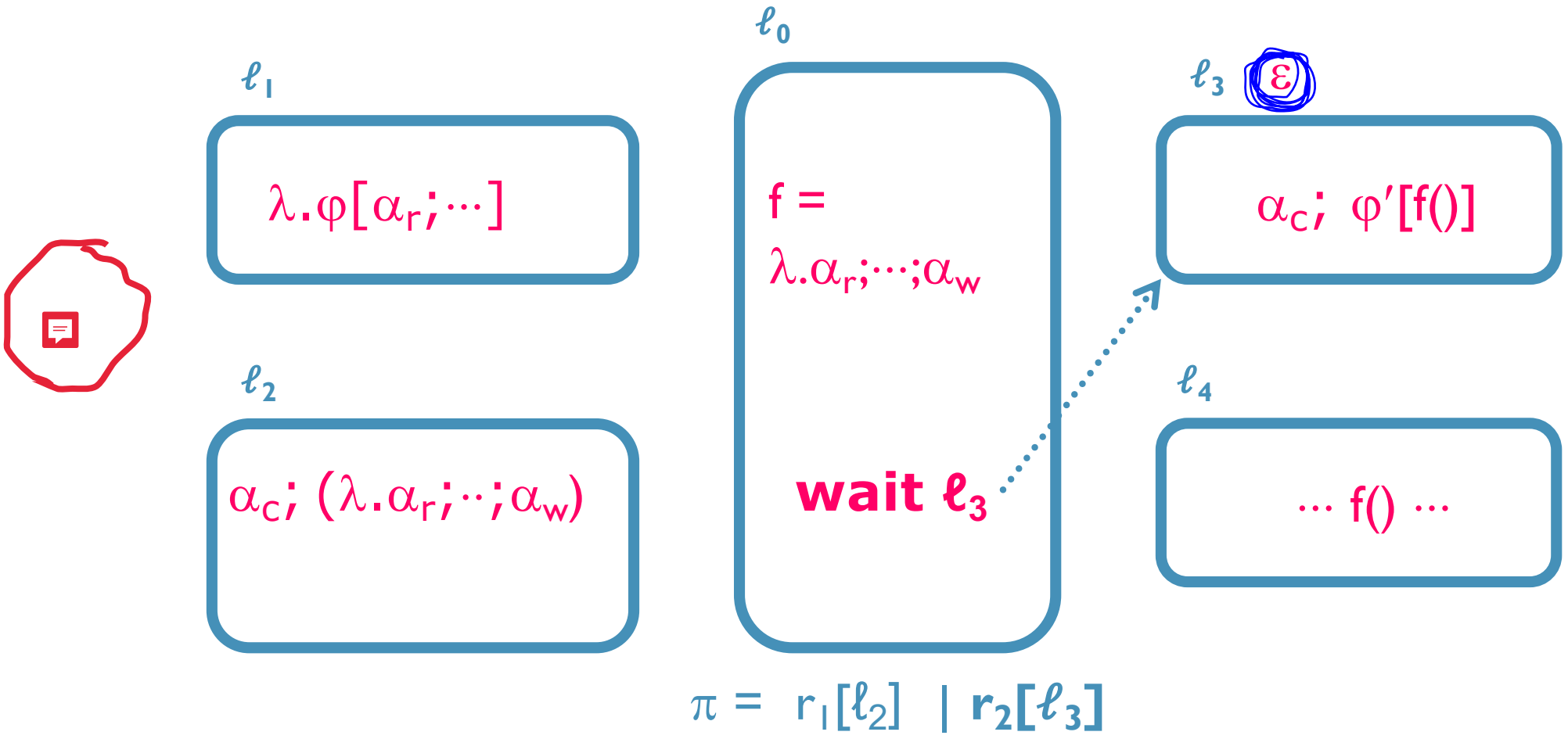
EXECUTING A NETWORK OF SERVICES



EXECUTING A NETWORK OF SERVICES

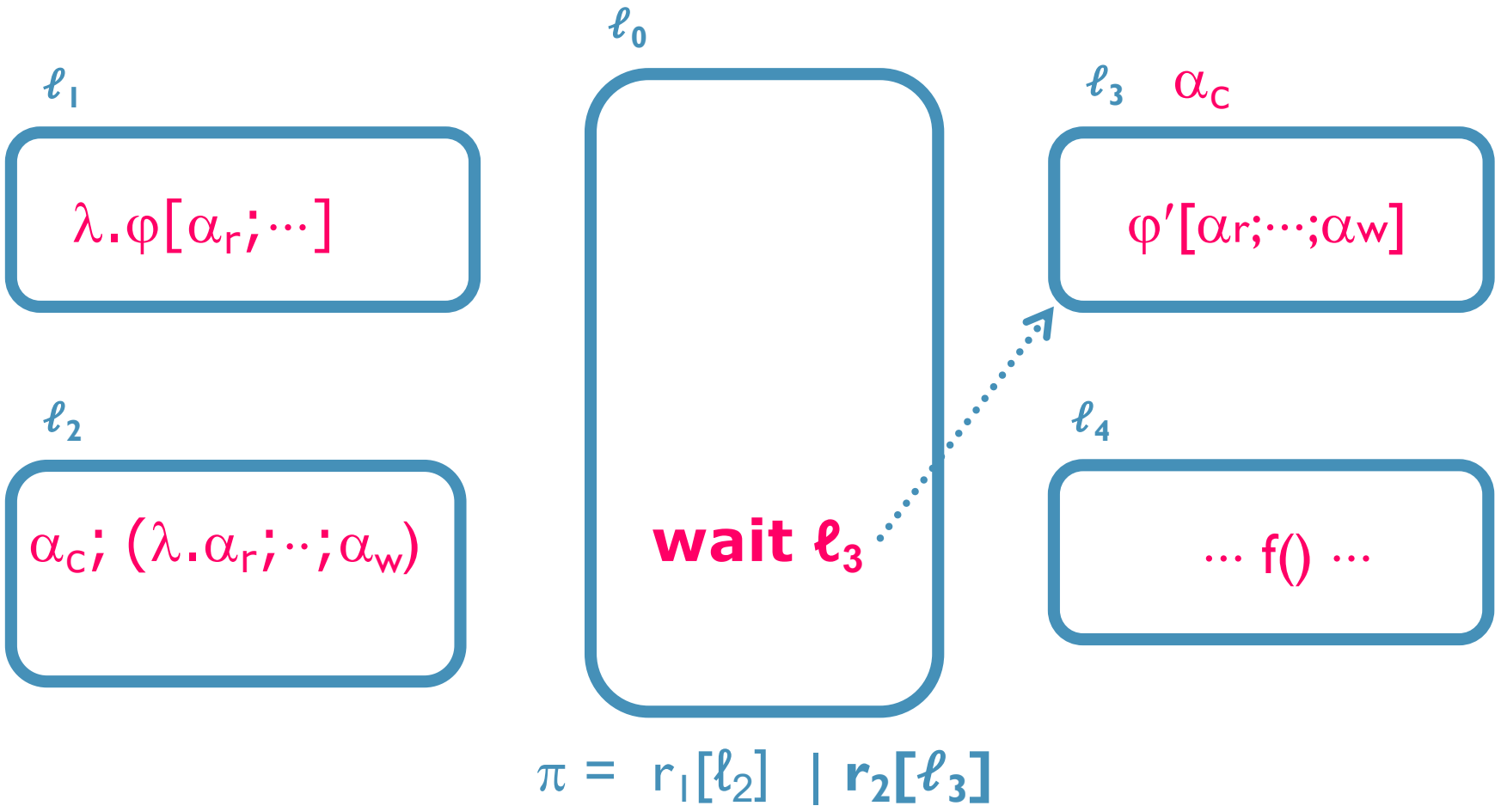


EXECUTING A NETWORK OF SERVICES



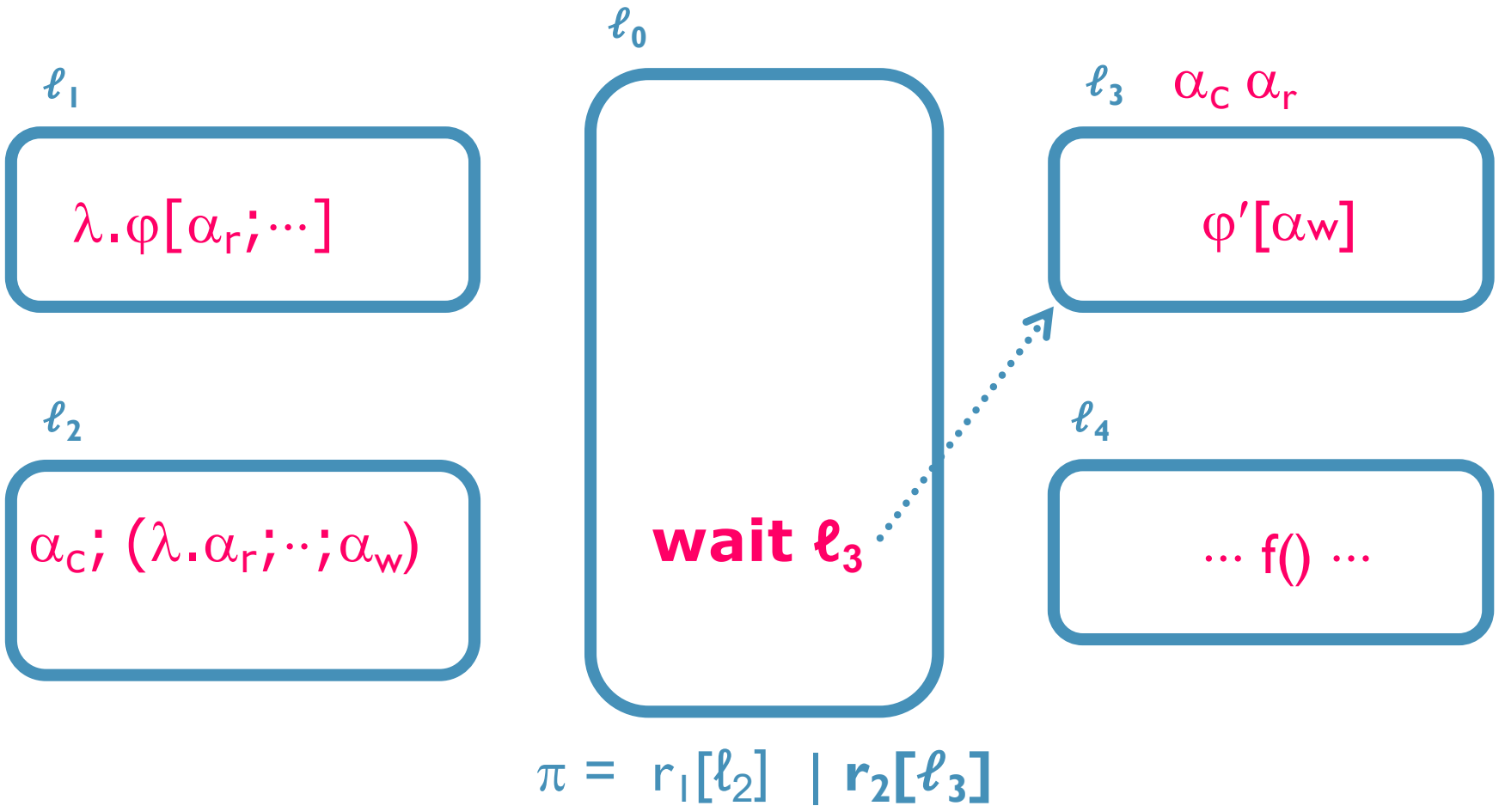
La politica mi dice che io non posso eseguire una write dopo aver fatto una read. Intanto l3 per l'esecuzione ha già stabilito che ci sono certificati.

EXECUTING A NETWORK OF SERVICES

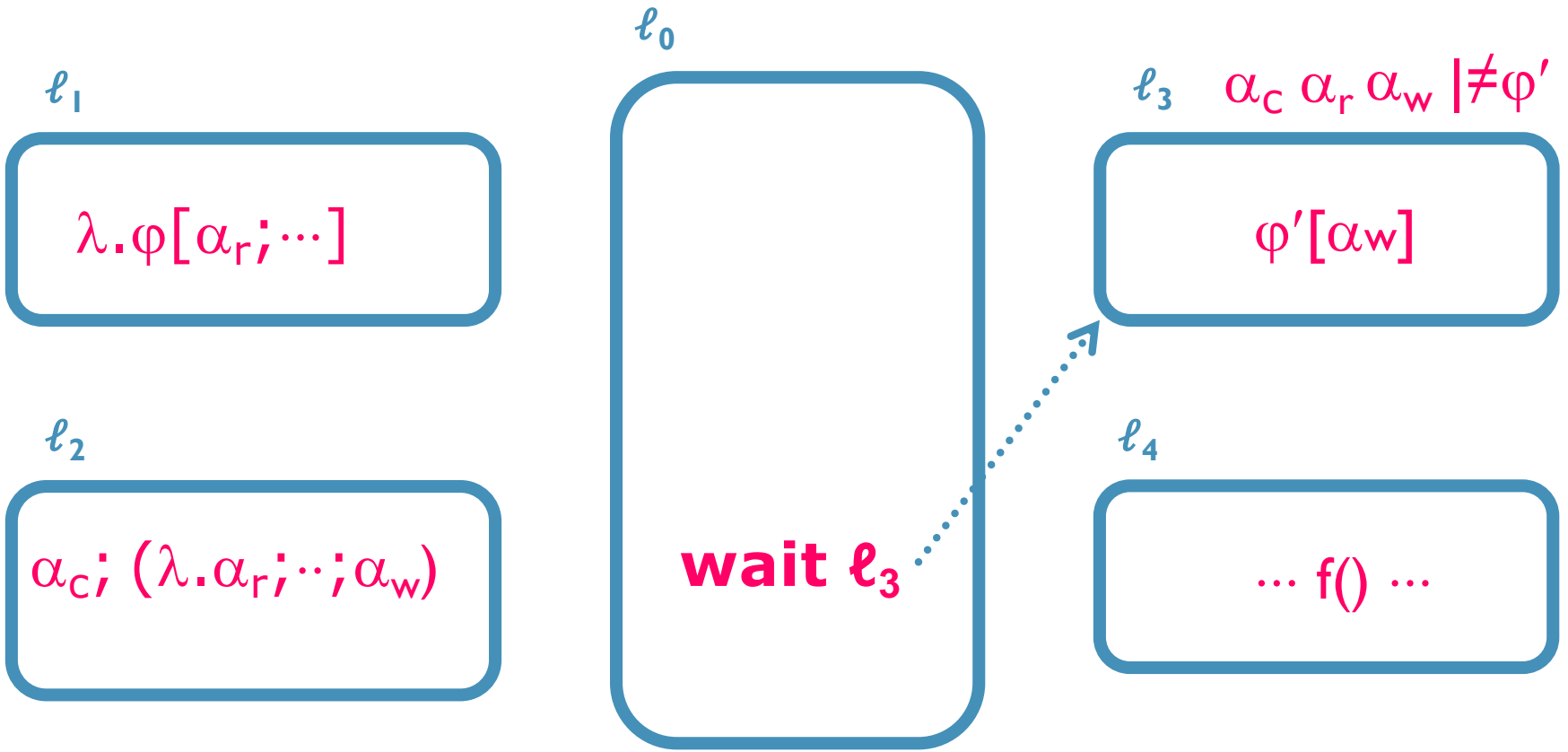


A questo punto proseguiamo nella esecuzione, eseguiamo un read

EXECUTING A NETWORK OF SERVICES



EXECUTING A NETWORK OF SERVICES



$\pi = r_1[\ell_2] \mid r_2[\ell_3]$ **not satisfied!**

FOCUS ON FUNCTIONS + SECURITY

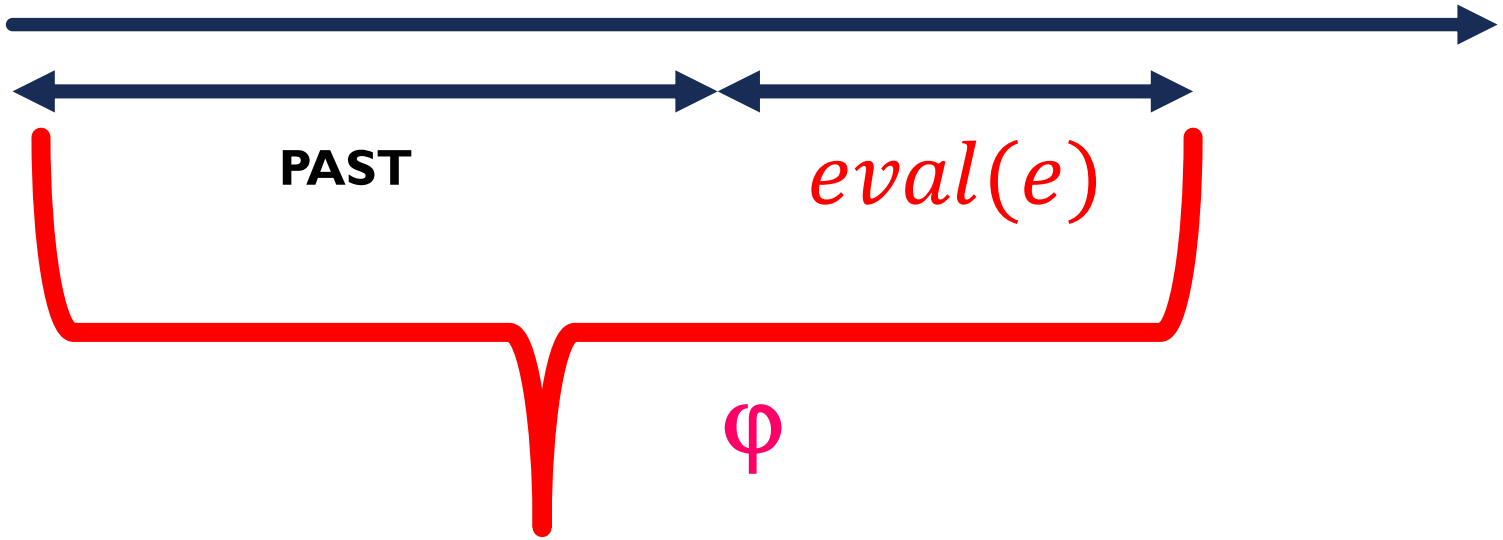
Services	e ::= x	variable
	α	access event
	if b then e else e′	conditional
	λ x.e	Lambda
	λ_z x.e	Recursive lambda
	e e′	application
	φ[e]	safety framing

Vorrei fare a un piccolo focus per cercare di capire un po' meglio alcune cose relativamente a cosa stiamo facendo quando mettiamo insieme un linguaggio funzionale con un safety framing. Per il momento mi focalizzo su questa parte. Questa parte, vedete, è un nucleo di linguaggio funzionale a cui ho inserito un framing che va semplicemente a controllare delle proprietà di safety relative all'esecuzione del codice che viene monitorato.

Allora andiamo a rappresentare graficamente che cosa vuol dire PHI di e. Abbiamo detto che tutta la storia dell' esecuzione è caratterizzata da degli eventi. Bene, allora a questo punto questi eventi devono soddisfare la politica. Quindi se noi vediamo questa freccia più grande come l'esecuzione del programma, quando noi mandiamo in esecuzione PHI di e, stiamo facendo le valutazioni di e, e tutto questo deve essere non solo la porzione che corrisponde alla valutazione dell'espressione che deve soddisfare la politica PHI ma tutto, anche il passato. Cioè quando io vado a valutare una espressione sotto il controllo di una politica, non considero gli eventi di sicurezza che mi sono generati dalle esecuzione di e, ma considero tutto quello che è successo fino a quel punto nell'esecuzione del mio programma. Allora, vedete, questo è leggermente diverso rispetto al meccanismo che noi abbiamo già visto, ovvero la stack inspection. Quando voi mandate in esecuzione un metodo, cosa andate a vedere per vedere se nel meccanismo della stack inspection quel metodo ha i diritti per essere eseguito? Andate a vedere lo stack. Lo stack rappresenta il flusso di esecuzione e infatti, quello che vi aspettate è che il metodo sulla testa dello stack ha diritto di eseguire una certa operazione se tutti i frame quindi tutti i record di attivazione dei metodi ancora attivi hanno lo stesso diritto. Ma ribadisco, andate a vedere sullo stack se tutti i metodi ancora attivi hanno lo stesso diritto. Questo meccanismo non considera ad esempio, il fatto che ci potrebbero essere stati dei metodi che sono stati eseguiti nel passato e che hanno fatto delle operazioni. Perché non le considera? Perché questi metodi potrebbero aver restituito il controllo e se hanno restituito il controllo, non sono più presenti sulla stack e quindi non le va a controllare. Questo meccanismo invece definisce sia una politica locale che ha a che vedere con la valutazione, ma per definire le caratteristiche della politica locale va a vedere non solo quello che è presente sullo stack, ma quello che ci è stato nel passato. Ovvero va a vedere anche dei flussi di informazione implicita che potrebbero essere stati generati durante esecuzioni precedenti.

HISTORY DEPENDENT ACCESS CONTROL

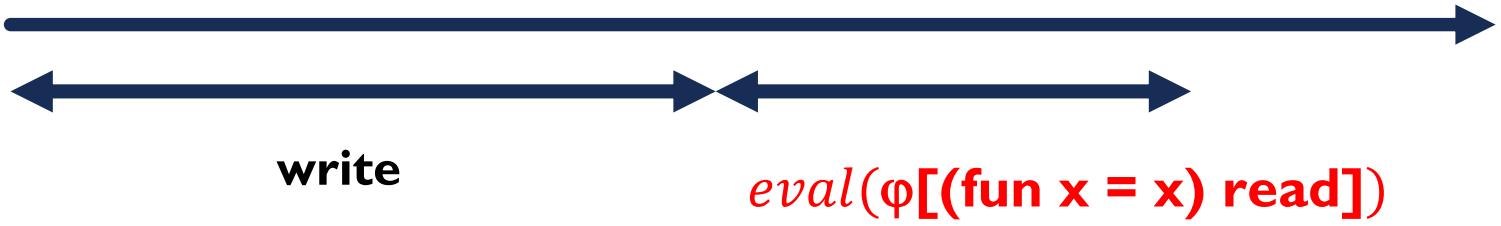
During the evaluation of $\varphi[e]$, the whole execution history (i.e. the past history followed by the events generated so far by e) must respect the policy φ.



EXAMPLE

write; φ[(fun x = x) read]

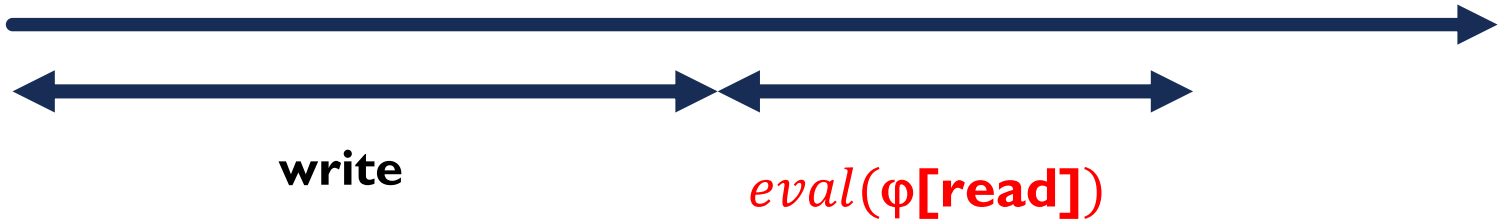
Policy φ = no read after write



EXAMPLE

write; φ[(fun x = x) read]

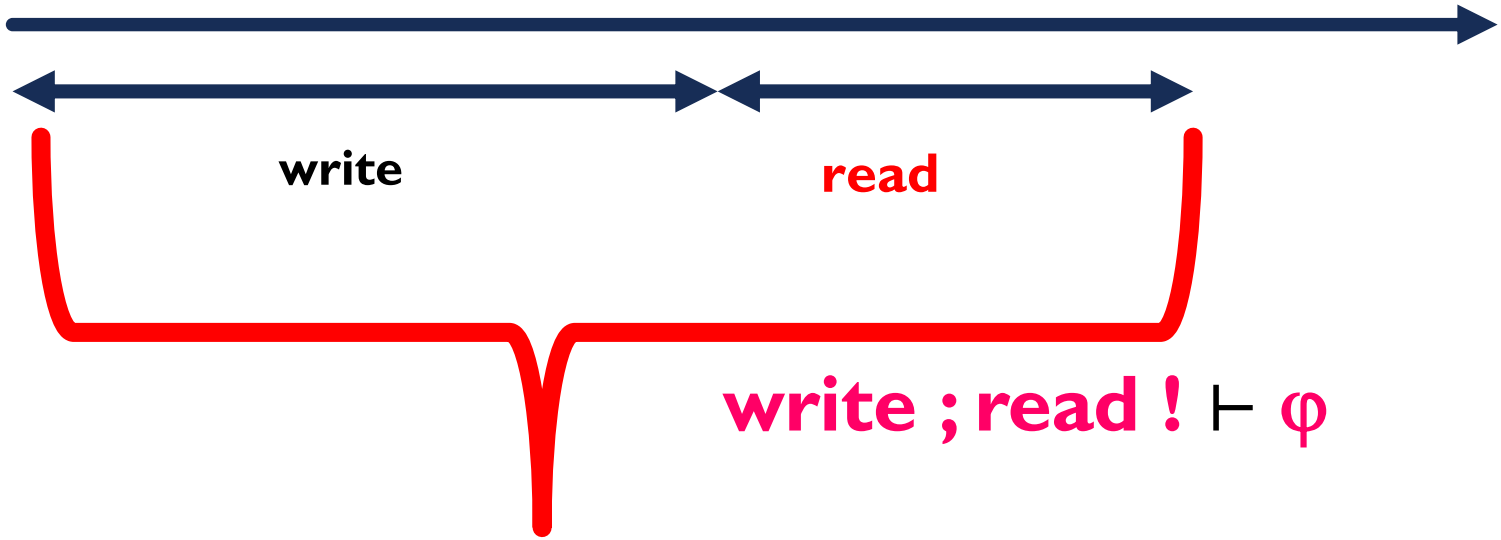
Policy φ = no read after write



Supponiamo di essere in questa operazione notate che sto utilizzando una meta notazione. Supponiamo di avere questo programma che fa un'operazione di scrittura write. Poi va a eseguire un'applicazione funzionale che è esattamente l'applicazione funzionale della funzione identità e un'operazione di lettura, quindi questo è un modo per scrivere che sta facendo una read all'interno di una composizione, quindi vuol dire che il risultato di quello che ho all'interno del frame è qualcosa di più articolato che ha a che vedere anche con esecuzione di funzioni e dell'applicazione di funzioni. Supponiamo che la politica PHI che qui ho definito è che non posso leggere dopo aver scritto, l'idea è che se io ho scritto informazioni locali poi non posso leggerla perché questa politica serve in un qualche modo per poter evitare che ci siano dei flussi di informazione implicita perche' se ho letto posso avere inserito poi i valori della lettura su delle variabili locali e posso utilizzare poi queste variabili locali per scrivere da altre parti, pertanto voglio evitare che ci sia questo flusso implicito. Allora sono nella solita freccia dell' esecuzione. Nel passato ho fatto una write adesso vado ad eseguire, quindi faccio l'eval e chiamano l'interprete con la funzione identità applicata alla read. Questo qui vuol dire che vado ad eseguire l'applicazione funzionale e vado a eseguire la read. A questo punto devo dire ma dove vale la politica? La politica vale localmente o vale complessivamente. Se vale localmente questa politica dato che la read è stata già eseguita e non è sullo stack quello che ho è che non vedrei write. Ma se leggo dell'informazione locale potrei leggere delle informazioni che ho ottenuto tramite una write. A questo punto invece il meccanismo che stiamo utilizzando dice di considerare non lo stack ma il log delle operazioni che sono state memorizzate sul execution history. Questo ci fa capire qual è la espressività di un meccanismo che cerca di amalgamare le politiche locali, perché questa PHI è una politica locale è un'execution monitor che va a controllare una porzione di codice con una nozione di flusso di esecuzione ma di flusso di esecuzione che tiene conto anche delle computazioni terminate. Quindi è diverso dalla stack inspection proprio per questo aspetto che tiene conto anche di quello che è successo nel passato ed è terminato.

write; φ[(fun x = x) read]

Policy φ = no read after write



AN EXAMPLE

We consider a simple web browser that displays HTML pages and runs mobile code (e.g. applets).

Applets can be trusted (e.g. because signed, or downloaded from a trusted site), or untrusted.

The browser has a site policy always applied to untrusted applets. The site policy says that an applet cannot connect to the web after it has read from the local disk

Supponiamo di avere programmato nel nostro linguaggio un qualcosa che è un browser che quello che fa è un display di pagine html poi può eseguire del codice mobile, ad esempio delle applet che possono essere certificate perché sono assegnate oppure possono essere non certificate, quindi untrusted. Il browser deve mettere delle politiche di sicurezza per andare a controllare l'esecuzione delle applet che scarica dalla rete. Allora assumiamo che il browser ha una politica globale complessiva che viene applicata a tutte le applet untrusted. E questa politica dice che: non puo' connettersi alla rete dopo aver letto informazione locale. Vuol dire che ha una politica che cerca di evitare che del codice mobile che viene eseguito localmente quando io do la proprietà di esecuzione sta dando una bella proprietà di sicurezza, può leggere, ma tutte le informazioni di lettura che ha fatto dalla lettura locale non le può rimandare fuori, quindi non posso avere un flusso di informazione che mi permette di sbattere in rete dell'informazione letta in locale.

TRUSTED APPLETS

Read = λ . αr \\ to read files,
let Read = ar

Write = λ . αw \\ to write files,
let Write = aw

Connect = λ . αc \\to open web connections
let Connect = ac

La prima applet che è trusted semplicemente dice: leggo e quindi la faccio con la meta notazione lambda, funzione senza parametri che fa una lettura. La write ugualmente mentre la connect apre una connession web.

THE BROWSER

Il browser è una funzione che prende due parametri, il primo parametro è l'URL quindi in qualche modo la pagina sta per dire la pagina su cui deve fare il display. La pagina su cui deve fare il display deve essere una pagina che soddisfa la proprietà di essere una pagina html, quindi se è una pagina html allora fa il display della pagina, altrimenti potrebbe essere una applet che viene scaricata. Allora a questo punto se l'applet è trusted, allora che cosa fa il browser? La esegue con una politica che è il parametro della funzione del browser, in modo particolare la politica è una funzione che prende l'applet da eseguire e la va a incapsulare all'interno di una politica PHI'. Se invece è untrusted, esegue l'applet u che gli viene data ma all'interno della politica del browser. Ricordate il browser che ha sempre una politica. La politica del browser dice che qualunque cosa che lui esegue untrusted può leggere localmente, ma appena cerca dopo aver letto di fare qualunque operazione di connessione in rete lo blocca, solleva una security exception. Poi a un certo punto fa un'operazione di write. Il fatto che li trovate una stellina è un modo per scrivere che non ha parametri, sennò si confondevano le varie operazioni.

Browser = λu.
λp. If html(u) then display(u)
else if trusted(u) then p u
else φ[p u; Write ∗]

let Browser = fun u =
fun p If html(u) then display(u)
else if trusted(u) then p u
else φ[p u; Write ∗]

Parameter p is a takes thr form of a function
p = λx. φ′[x∗]

Scriviamo un untrusted applet. Prende un una funzione zeta, una politica che è la politica identità quindi non mette alcuna politica di fatto, quindi è una politica estremamente liberale e poi prende esattamente il codice del browser in qualche modo ha al suo interno una chiamata al codice che si fa passare come meccanismo di spoofing per il browser in cui cerca di eseguire una applet con una politica vuota. L'identità è sostanzialmente: esegue la y senza applicare alcuna framing e quindi vuol dire che non ha una politica. Però dal punto di vista del tipo e' corretto. Quindi stiamo cercando di eseguire un' untrusted applet che fa delle operazioni di spoofing sull'applet.

AN UNTRUSTED APPLET

Untrusted = λz.λ .Browser z (λy.y∗)

let Untrusted = fun z Browser =
Browser z (fun y= y *)

The untrusted applet tries to spoof the browser by executing a supplied applet z with a void user policy

A RUN

Invochiamo il browser con l'applet untrusted a cui gli diamo Write e poi gli diamo la politica PHI' che è la politica locale. Succede che noi al browser che cosa gli passiamo? Gli passiamo l'untrusted write, che è il browser in cui gli passa il write con la void policy. Ancora una volta continua a rimanere epsilon cioè, non faccio alcuna operazione sull'uso delle risorse di fatto, quello che faccio , ho controllato se è un html, No, non è un html, è trusted, no, esattamente untrusted. Quindi se untrusted cosa devo andare a fare? Facendo tutte le operazioni di sostituzione del codice all'interno di questo browser, deve andare a eseguire la politica del sito PHI all'interno con la politica supplied dall'applet untrusted che sto considerando e poi eseguo. Quindi sostanzialmente cosa sta eseguendo? Sto eseguendo una scrittura lo esegue in una situazione dove ho prima la politica fornita, quella void e poi quella PHI che è la politica del sito del browser.

The run of an untrusted write, executed with a user policy φ′ saying that applets cannot write the local disk

$$
\begin{aligned}
& \varepsilon,\ Browser\,(Untrusted\ Write)\,(\lambda y.\varphi'[y*]) \\
\rightarrow^* \ & \varepsilon,\ \varphi[(\lambda y.\varphi'[y*])\,(\lambda\_.\,Browser\ Write\,(\lambda y.\,y*))];\ Write*] \\
\rightarrow^* \ & \varepsilon,\ \varphi[\varphi'[Browser\ Write\,(\lambda y.\,y*)];\ Write*] \\
\rightarrow^* \ & \varepsilon,\ \varphi[\varphi'[(\lambda y.\,y*)\,Write];\ Write*] \quad \rightarrow^* \quad \varepsilon,\ \varphi[\varphi'[\alpha_w];\ Write*]
\end{aligned}
$$

Browser = λu.
λp. If html(u) then display(u)
else if trusted(u) then p u
else φ[p u; Write ∗]

THE SECURITY EXCEPTION

$$\varepsilon,\ \varphi[\varphi'[\alpha_w]; Write*]$$

$$\boldsymbol{\alpha w\ !} \vdash\ \boldsymbol{\varphi'}$$

Allora a questo punto quello che succede è che io sto eseguendo la W, ho il PHI' e la politica. e sto facendo un'operazione di write quindi non mi soddisfa la politica e a quel punto solleva un'eccezione di sicurezza.

A SECOND RUN

Possibilmente potremmo anche fare la stessa operazione dove sto cercando di connettermi. Quindi vedete, vado ad eseguire sempre PHI e PHI' con Connect e a questo punto vedete quindi ho fatto prima un'operazione di lettura e poi mi connetto ma questo qui mi soddisfa la politica PHI' ma non mi soddisfa la politica esterna e quindi sollevo un'eccezione di sicurezza.

The untrusted applet that reads the local disk and then tries to connect.

$$
\begin{aligned}
& \varepsilon,\ Browser\ (Untrusted\ (\lambda\_.\ Read*;\ Connect*))\ (\lambda y.\varphi'[y*]) \\
\rightarrow^* \ & \varepsilon,\ \varphi[(\lambda y.\varphi'[y*])\ (Untrusted\ (\lambda\_.\ Read*;\ Connect*));\ Write*] \\
\rightarrow^* \ & \varepsilon,\ \varphi[\varphi'[Browser\ (\lambda\_.\ Read*;\ Connect*)\ (\lambda y.\ y*)];\ Write*] \\
\rightarrow^* \ & \varepsilon,\ \varphi[\varphi'[Read*;\ Connect*];\ Write*] \quad \rightarrow^* \quad \alpha_r,\ \varphi[\varphi'[Connect*];\ Write*]
\end{aligned}
$$

Browser = λu.
λp. If html(u) then display(u)
else if trusted(u) then p u
else φ[p u; Write ∗]

A SECURITY EXCEPTION

$$\alpha_r,\ \varphi[\varphi'[Connect*];\ Write*]$$

$$\boldsymbol{\alpha r\ \alpha c\ !} \vdash\ \varphi$$

A THIRD RUN

A questo punto faccio una terza esecuzione che vedete sta facendo un untrusted con una read e vedete che continua a passare questa politica PHI' e vedete che anche in questo caso nella esecuzione faccio una read e una write e da questo punto ho una politica sul vuoto. Bene, a questo punto l'operazione è eseguita perché lo scope della politica PHI' è esaurito. A questo punto la politica non mi controlla piu' le esecuzioni, mi vale soltanto la politica PHI quella esterna, allora a quel punto la lambda è vero che è eseguita però a questo punto, non avendo fatto una connessione mi soddisfa la politica del sito è quindi è arrivata completamente. Quindi questo cosa vuol dire? Vuol dire che riusciamo a controllare con questi meccanismi di sicurezza. Riusciamo a controllare l'esecuzione di codice untrusted all'interno di applicazioni dove si riesce a segnare esattamente i check di sicurezza che possono violare le politiche.

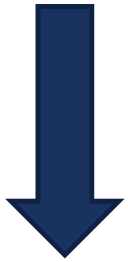
the untrusted read

$$
\begin{aligned}
& \varepsilon,\ Browser\,(Untrusted\ Read)\,(\lambda y.\varphi'[y*]) \\
\rightarrow^* \ & \varepsilon,\ \varphi[(\lambda y.\varphi'[y*])\,(Untrusted\ Read); Write*] \\
\rightarrow^* \ & \varepsilon,\ \varphi[\varphi'[Browser\ Read\,(\lambda y.\ y*)]; Write*] \ \rightarrow^* \varepsilon,\ \varphi[\varphi'[Read*]; Write*] \\
\rightarrow^* \ & \alpha_r,\ \varphi[\varphi'[*]; Write*] \ \rightarrow^* \ \alpha_r,\ \varphi[Write*] \ \rightarrow^* \ \alpha_r\alpha_w,\ \varphi[*]
\end{aligned}
$$

Browser = λu.
λp. If html(u) then display(u)
else if trusted(u) then p u
else φ[p u; Write ∗]

THE SECURITY CHECK

$$\alpha_r \alpha_w, \ \varphi[*]$$



$$\boldsymbol{\alpha r \ \alpha w \vdash \varphi}$$

The write operation is executed because the scope of the policy φ' has been left upon termination of the untrusted applet.

[App1]

$$\frac{\eta, e_1 \to \eta', e_1'}{\eta, e_1\ e_2 \to \eta', e_1'\ e_2}$$

[App2]

$$\frac{\eta, e_2 \to \eta', e_2'}{\eta, v\ e_2 \to \eta', v\ e_2'}$$

[AbsApp]

$$\eta, (\lambda_z x.e)\ v \to \eta, e\{v/x, \lambda_z x.e/z\}$$

[If]

$$\eta, \textbf{if b then } e_{\textbf{true}} \textbf{ else } e_{\textbf{false}} \to \eta, e_{\mathcal{B}(b)}$$

[Event]

$$\eta, \alpha \rightarrow \eta\alpha, ()$$

[Framing In]

$$\frac{\eta, e \rightarrow \eta', e' \qquad \eta' \models \varphi}{\eta, \varphi[e] \rightarrow \eta', \varphi[e']}$$

[Framing Out]

$$\frac{\eta \models \varphi}{\eta, \varphi[v] \rightarrow \eta, v}$$

SEMANTICS OF NETWORKS (1)

[Inject]

$$\frac{\eta, e \rightarrow \eta', e'}{\ell: \eta, e \rightarrow_{\pi} \ell: \eta', e'}$$

[Par]

$$\frac{N_1 \rightarrow_{\pi} N_1'}{N_1 \,||\, N_2 \rightarrow_{\pi} N_1' \,||\, N_2}$$

SEMANTICS OF NETWORKS (2)

[Request] $\pi = r[\ell'] \mid \pi'$

$$\ell : \eta, \mathbf{req}_r\ v \;\|\; \ell'\{e'\} : \varepsilon, \bigstar \;\rightarrow_{\pi}\; \ell : \eta, \mathbf{wait}\ \ell' \;\|\; \ell'\{e'\} : \varepsilon, e'v$$

[Reply]

$$\ell : \eta, \mathbf{wait}\ \ell' \;\|\; \ell'\{e'\} : \eta', v \;\rightarrow_{\pi}\; \ell : \eta, v \;\|\; \ell'\{e'\} : \varepsilon, \bigstar$$

OTHER KINDS OF PLANS

- **Simple plans** $\pi ::= 0 \mid \pi \mid \pi \mid r[\ell]$

$$\ell : \mathrm{req}_r \;||\; \ell' : \{P\} \rightarrow_{r[\ell']} \ell : \mathrm{wait}\ \ell' \;||\; \ell' : P$$

- **Multi-choice plans** $\pi ::= 0 \mid \pi \mid \pi \mid r[\ell_1 \ldots \ell_k]$

$$\ell : \mathrm{req}_r \;||\; \ell' : \{P\} \rightarrow_{r[\ell', \ell'']} \ell : \mathrm{wait}\ \ell' \;||\; \ell' : P$$

- **Dependent plans** $\pi ::= 0 \mid \pi \mid \pi \mid r[\ell.\, \pi]$

$$\ell : r[\ell'.\, \pi] \triangleright \mathrm{req}_r \;||\; \ell' : \{P\} \rightarrow \ell : r[\ell'.\, \pi] \triangleright \mathrm{wait}\ \ell' \;||\; \ell' : \pi \triangleright P$$

- …many others: multi+dependent, regular, dynamic,…

RECAP

- calculus: syntax and **operational semantics**
  - **Service as a function**
  - **call by properties**
  - **Operational semantics & orchestration**