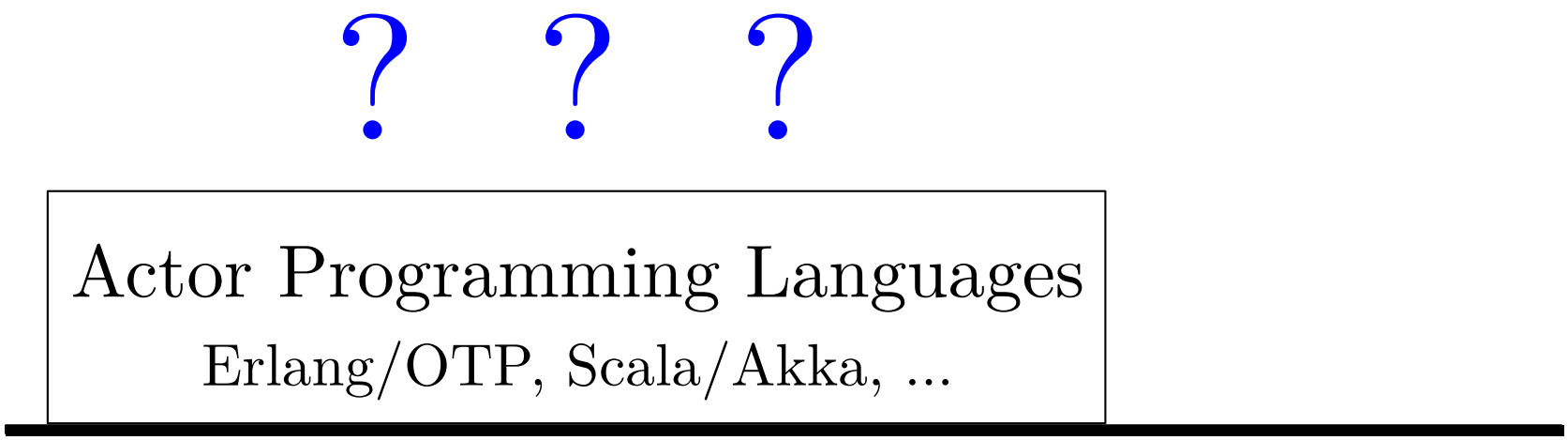
ESOP 2014, Grenoble, France; 20140410

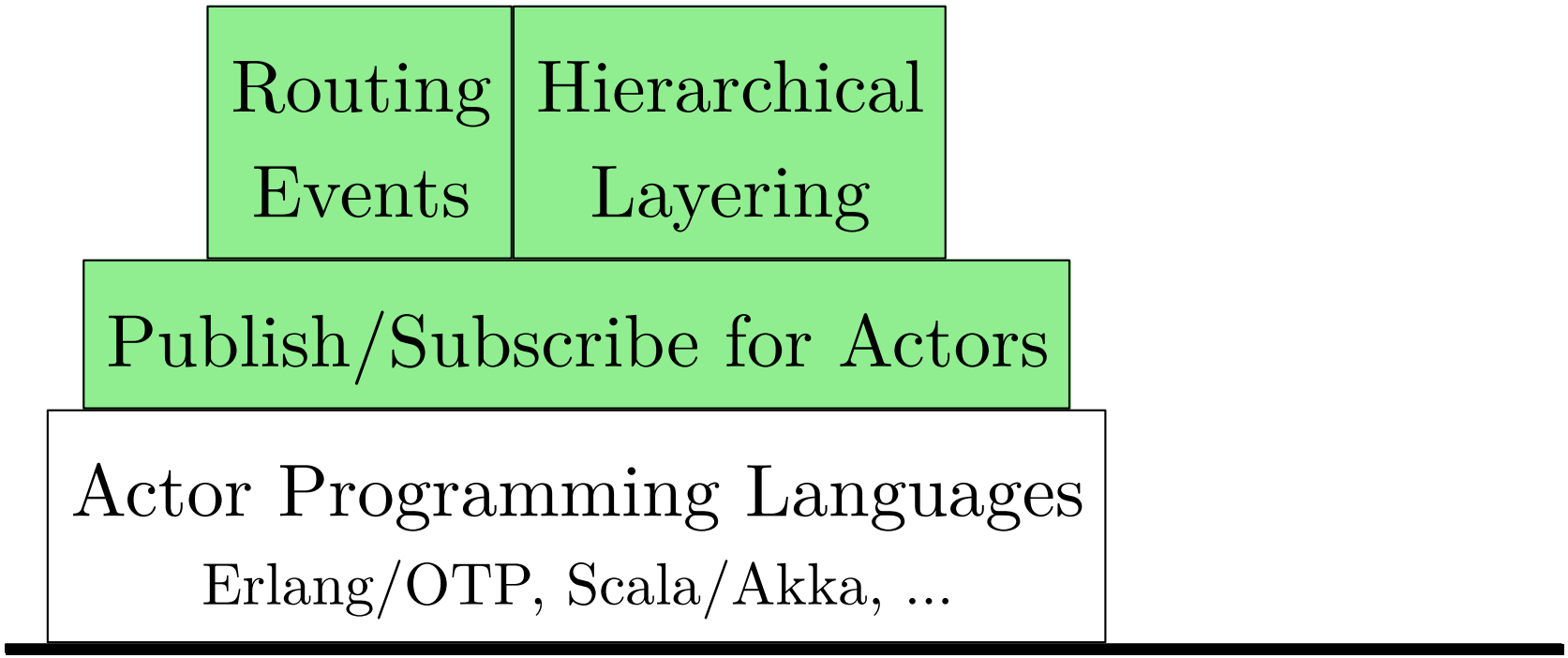
The Network as a Language Construct

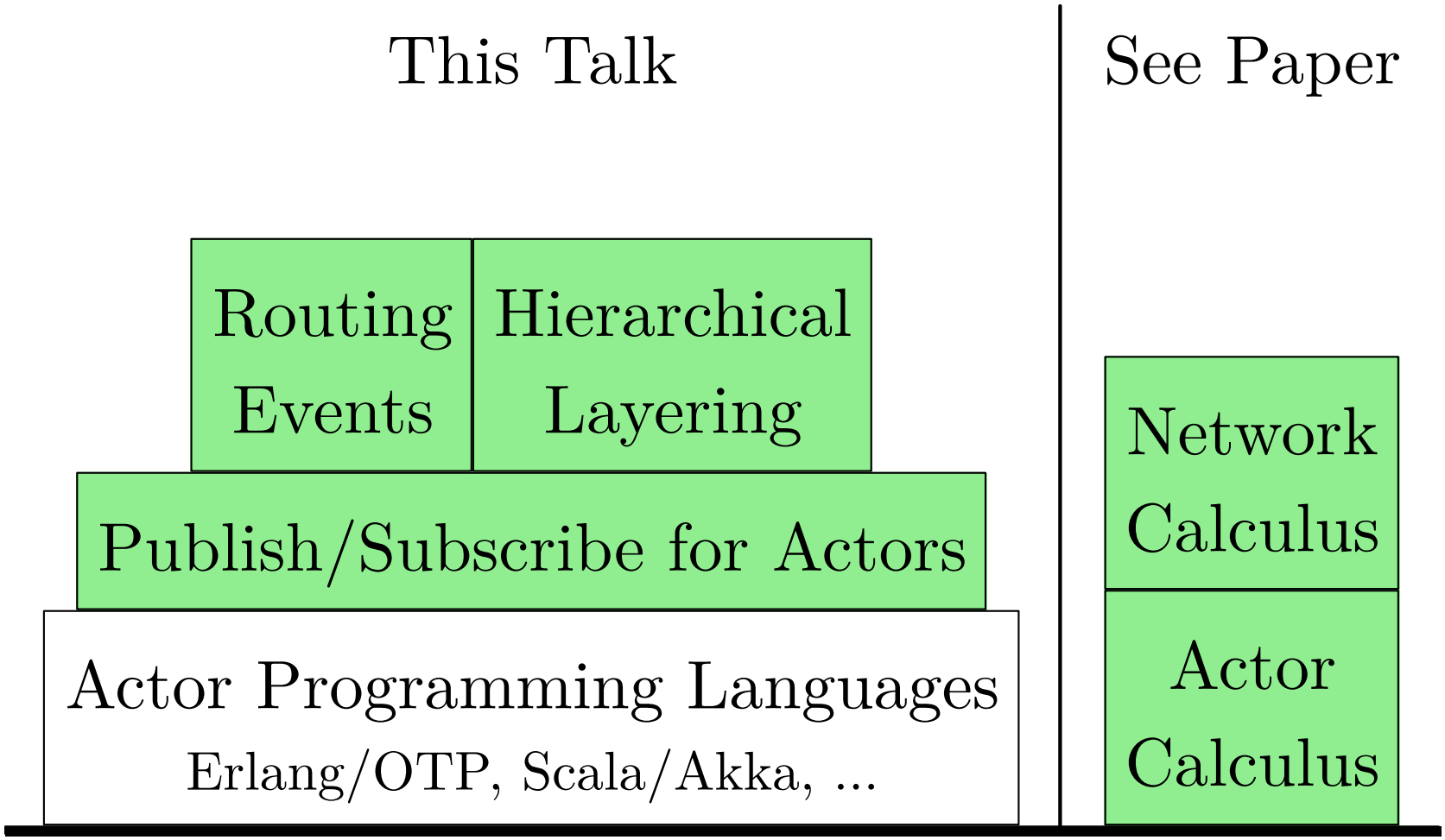
Tony Garnock-Jones Sam Tobin-Hochstadt Matthias Felleisen

Actor Programming Languages

Erlang/OTP, Scala/Akka, ...


? ? ?
Actor Programming Languages
Erlang/OTP, Scala/Akka, ...

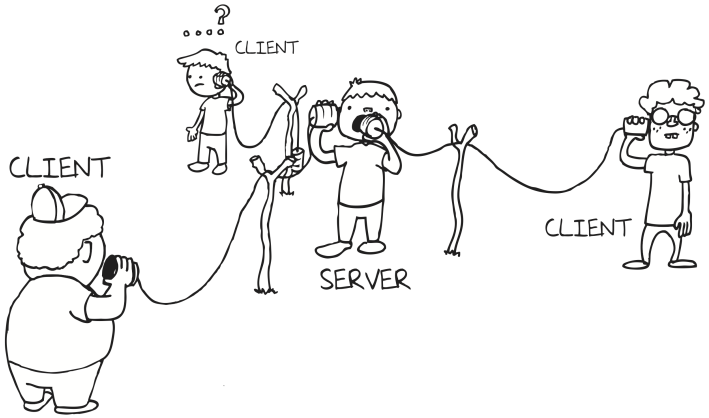

Routing
Events
Hierarchical
Layering
Publish/Subscribe for Actors
Actor Programming Languages
Erlang/OTP, Scala/Akka, ...


This Talk
See Paper
Routing Events
Hierarchical Layering
Publish/Subscribe for Actors
Actor Programming Languages
Erlang/OTP, Scala/Akka, ...
Network Calculus
Actor Calculus

PART I: The Problem

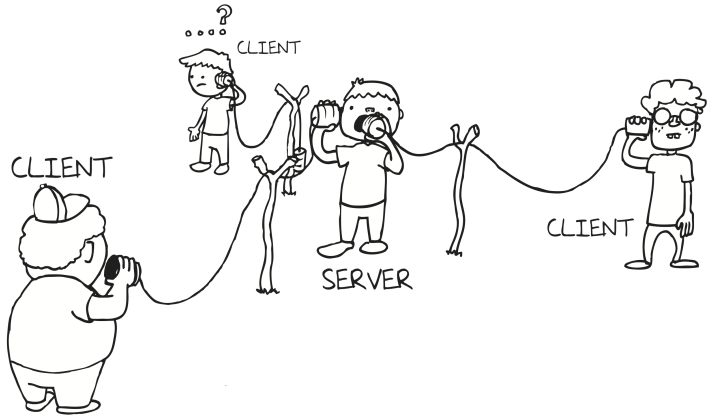
Functional I/O

Scaling up **big-bang** from domain-specific to general functional I/O



Functional I/O

Scaling up **big-bang** from domain-specific to general functional I/O

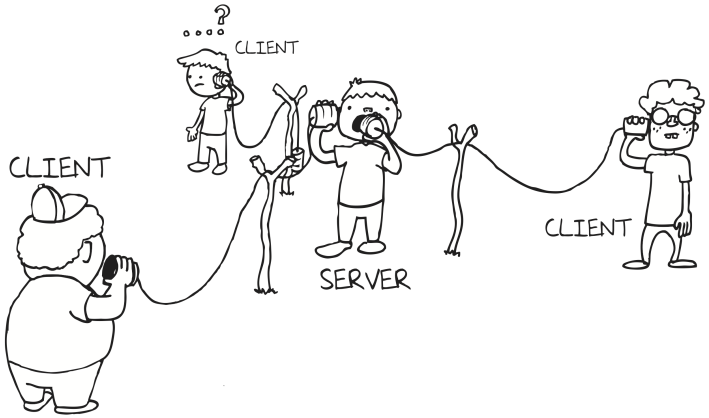


Apps in a functional I/O style:

- echo server
- multi-user chat
- DNS server
- SSH server

Functional I/O

Scaling up **big-bang** from domain-specific to general functional I/O

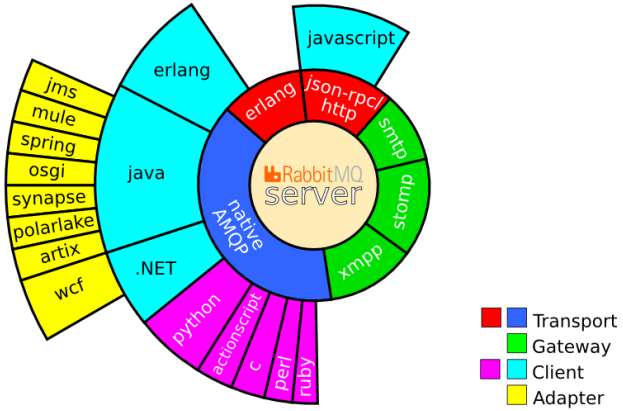


Apps in a functional I/O style:

- echo server
- multi-user chat
- DNS server
- SSH server

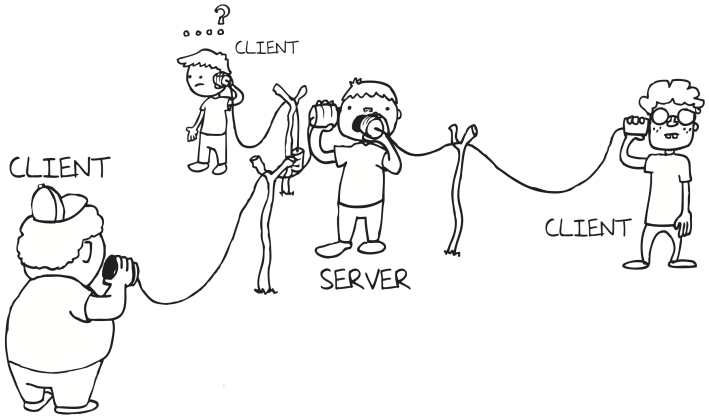
Distributed Systems

Implementing **RabbitMQ** and using it to build distributed systems



Functional I/O

Scaling up **big-bang** from domain-specific to general functional I/O

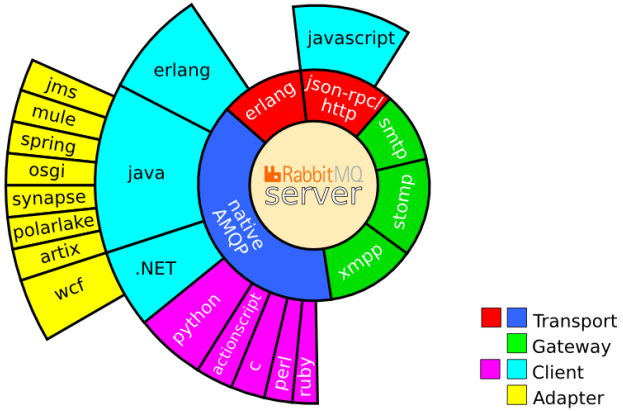


Apps in a functional I/O style:

- echo server
- multi-user chat
- DNS server
- SSH server

Distributed Systems

Implementing **RabbitMQ** and using it to build distributed systems



Investigated other paradigms:

- OO languages
- Network architecture
- CORBA services
- Erlang applications

Ubiquitous Patterns and Problems

Event broadcasting

Naming service

Service discovery

Startup ordering

Crash/exit signalling

Conversation management

Ubiquitous Patterns and Problems

Event broadcasting

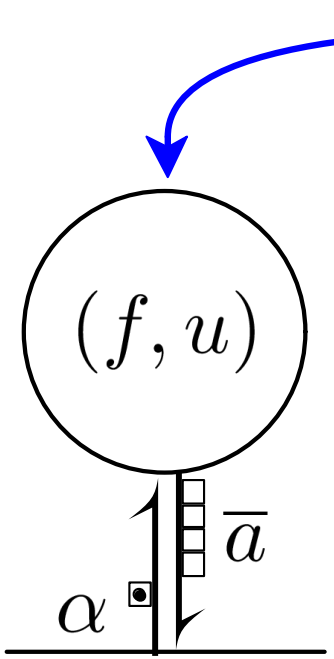
Naming service

Uniform Linguistic Solution

Crash/exit signalling

Conversation management

Recipe for Actor Languages



Base language actor

$$f(\alpha, u) = (\overline{a}, u')$$

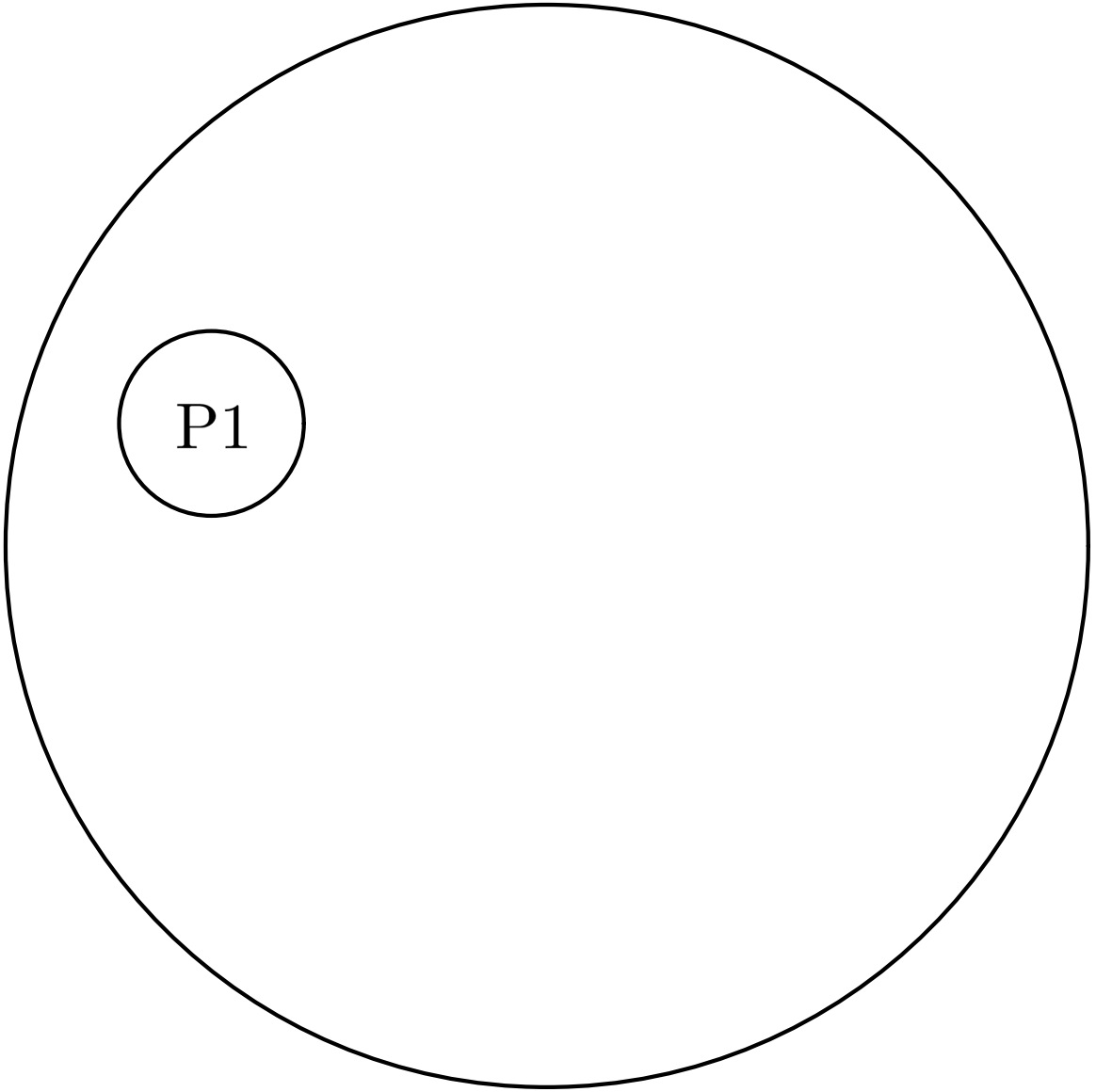
α = event, a = action

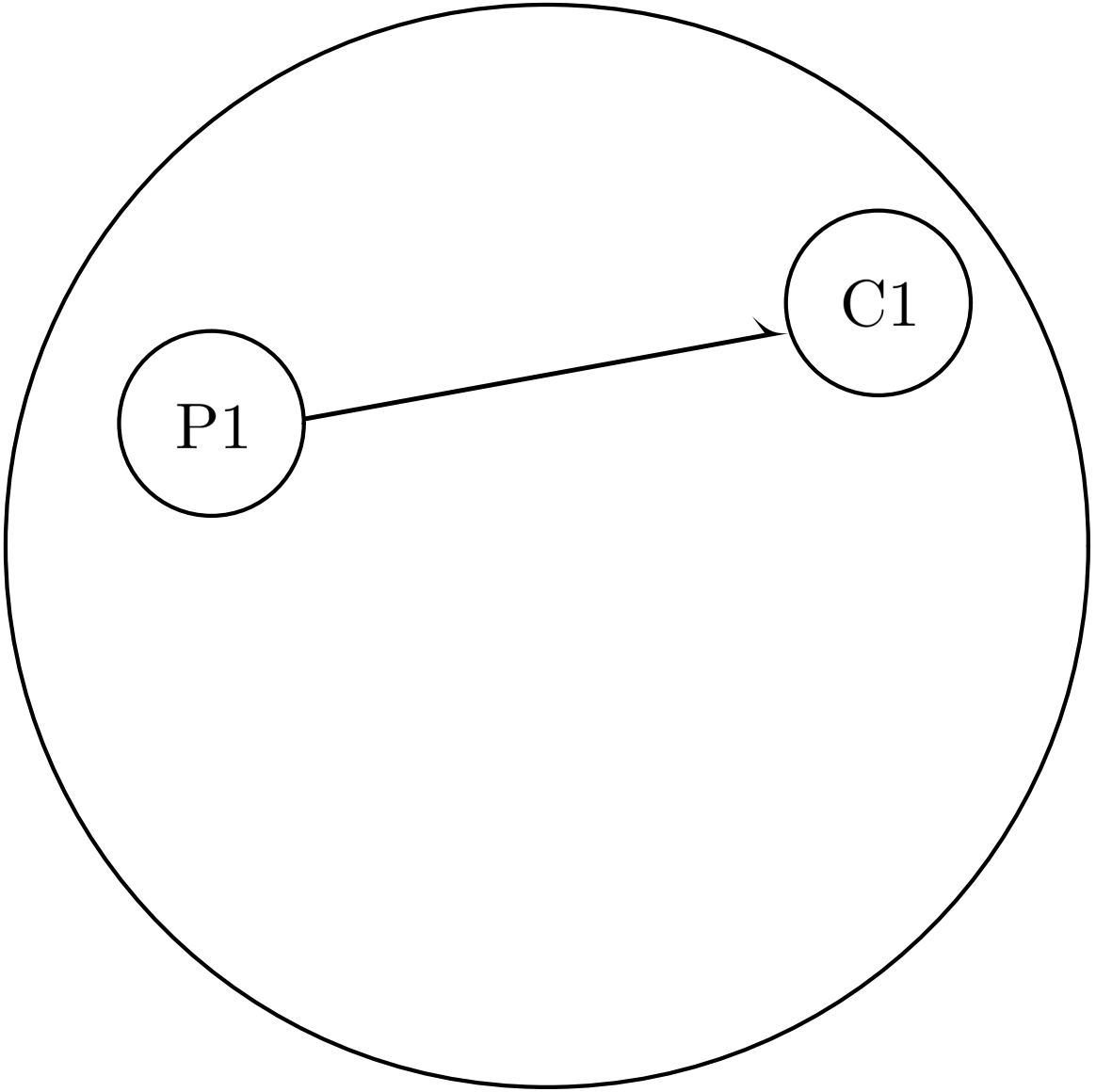
"Network"

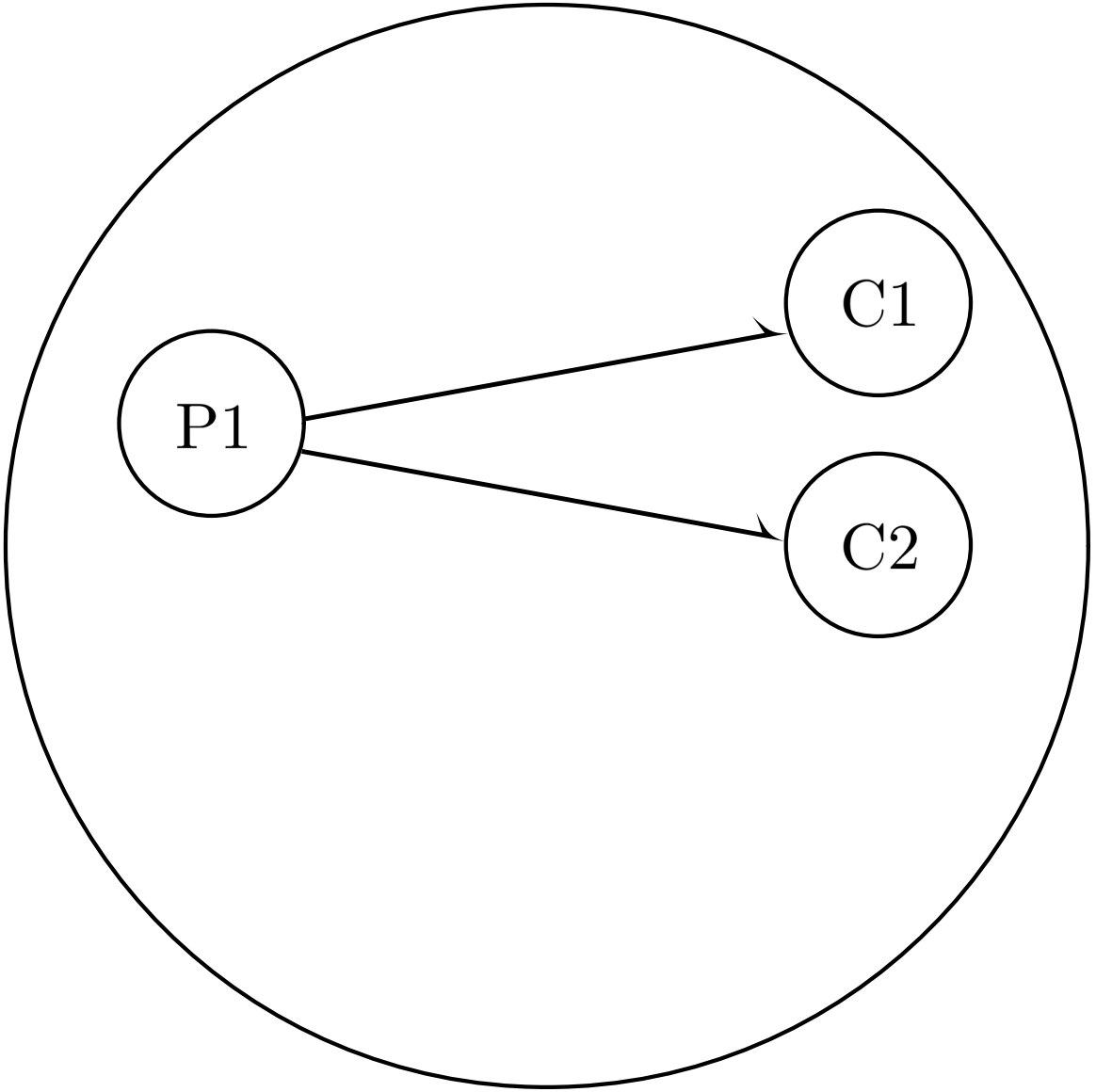
Log producers $\overset{\text{log messages}}{\longrightarrow}$ Log consumers

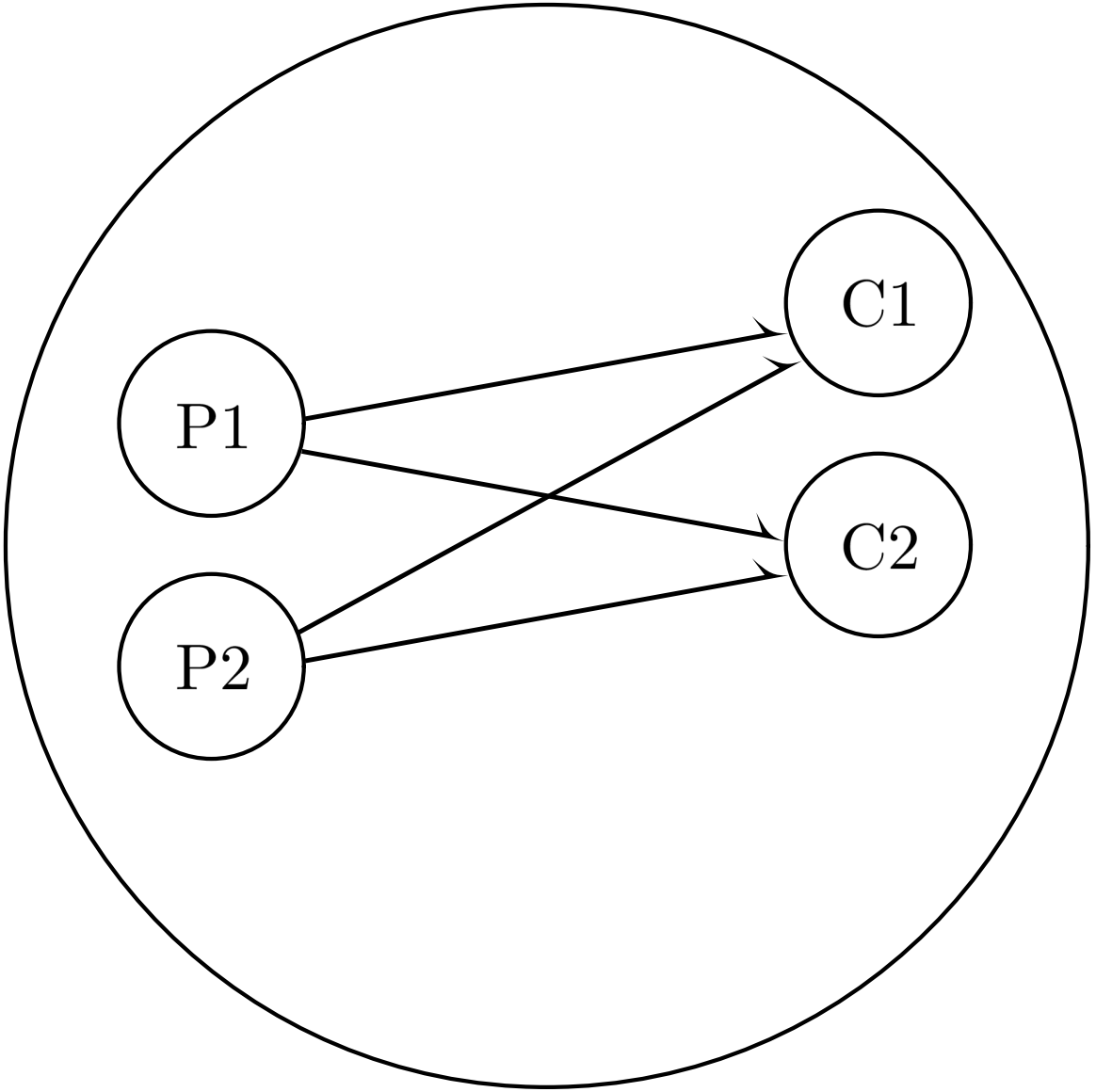
$\langle \mathsf{log}, [subsystem, severity, data] \rangle$

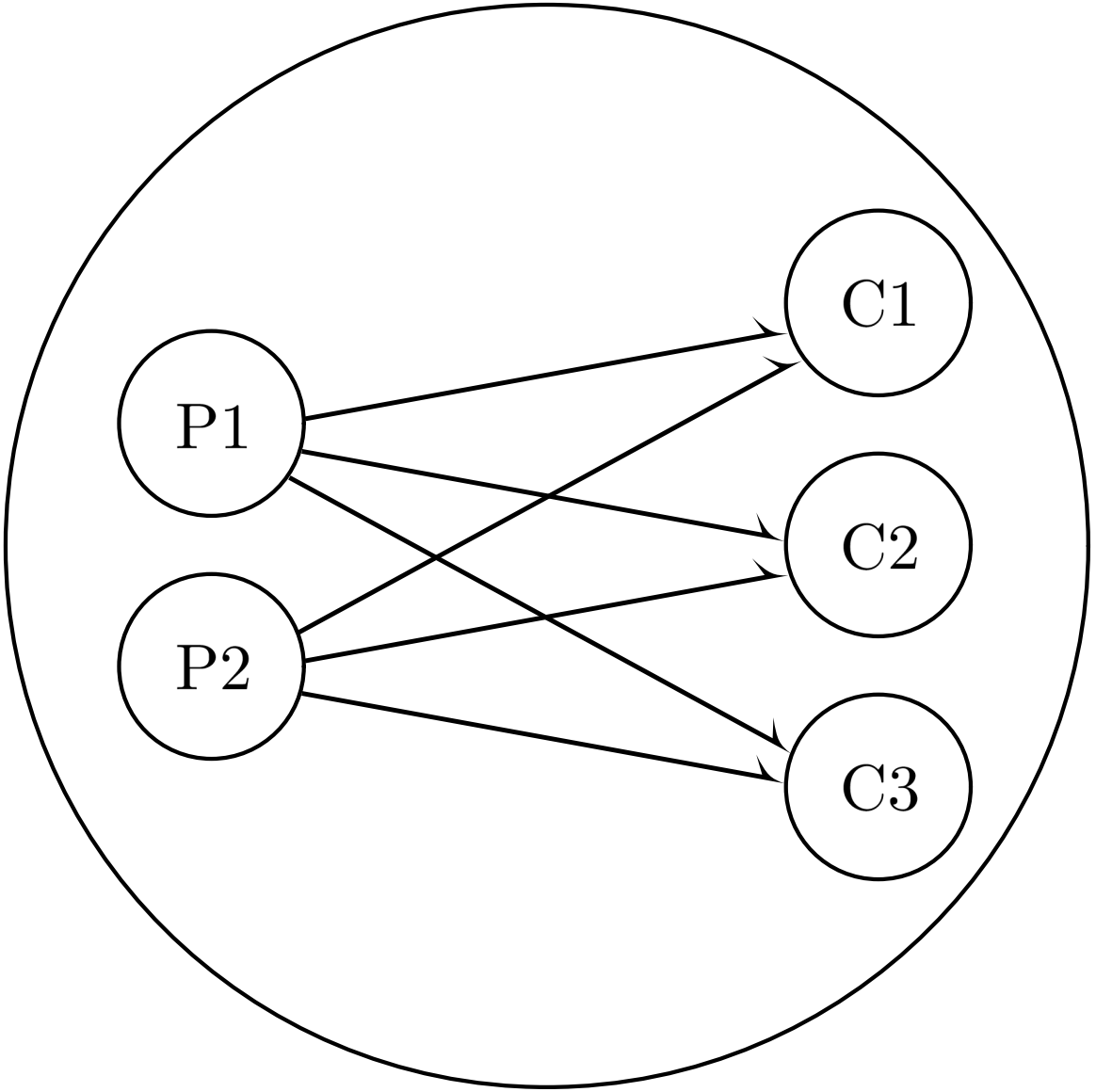
Consumers filter by subsystem, severity


P1


C1
P1


C1
P1
C2

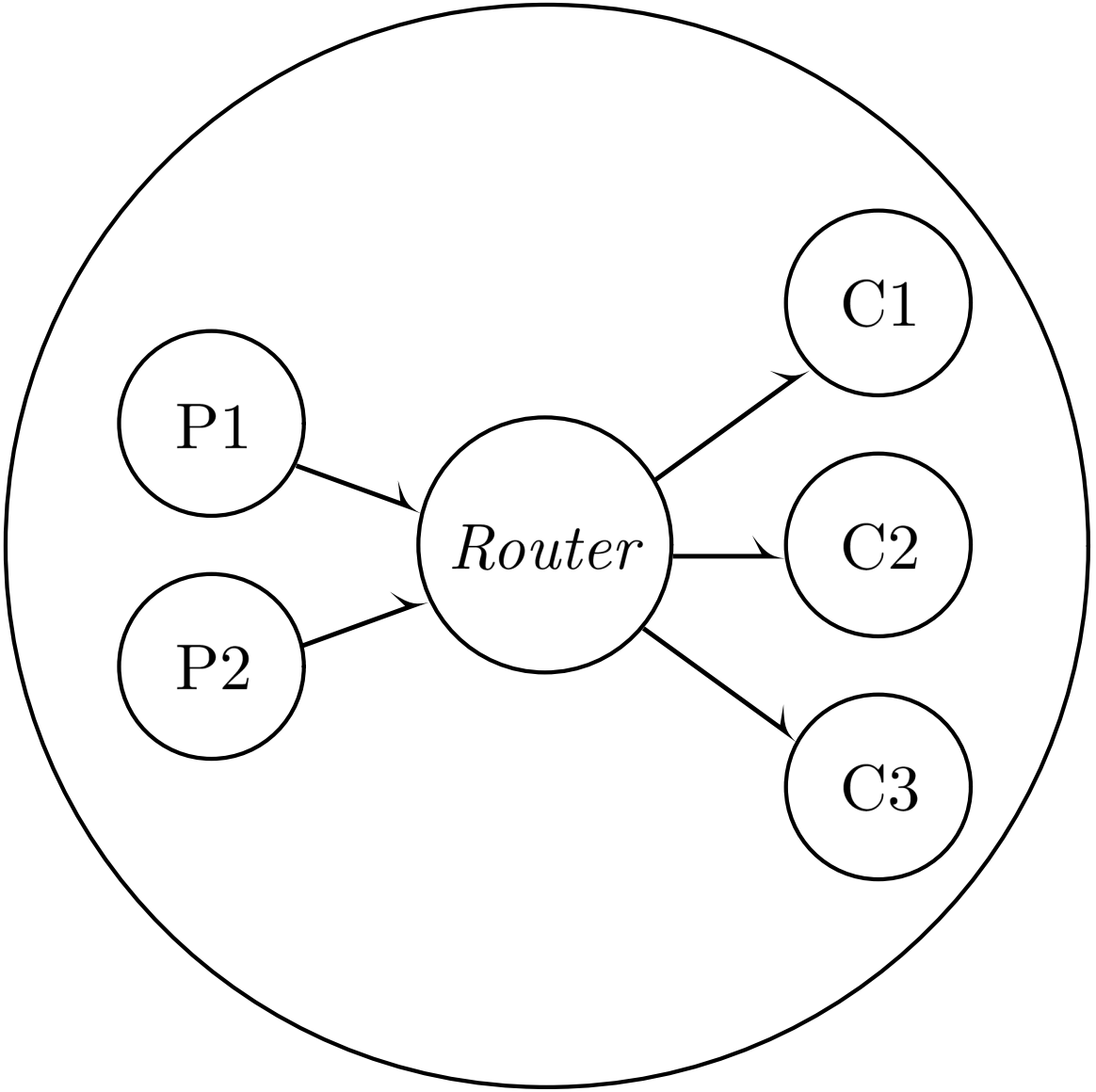

C1
P1
C2
P2


P1
P2
C1
C2
C3

Logging: Requirements Scorecard

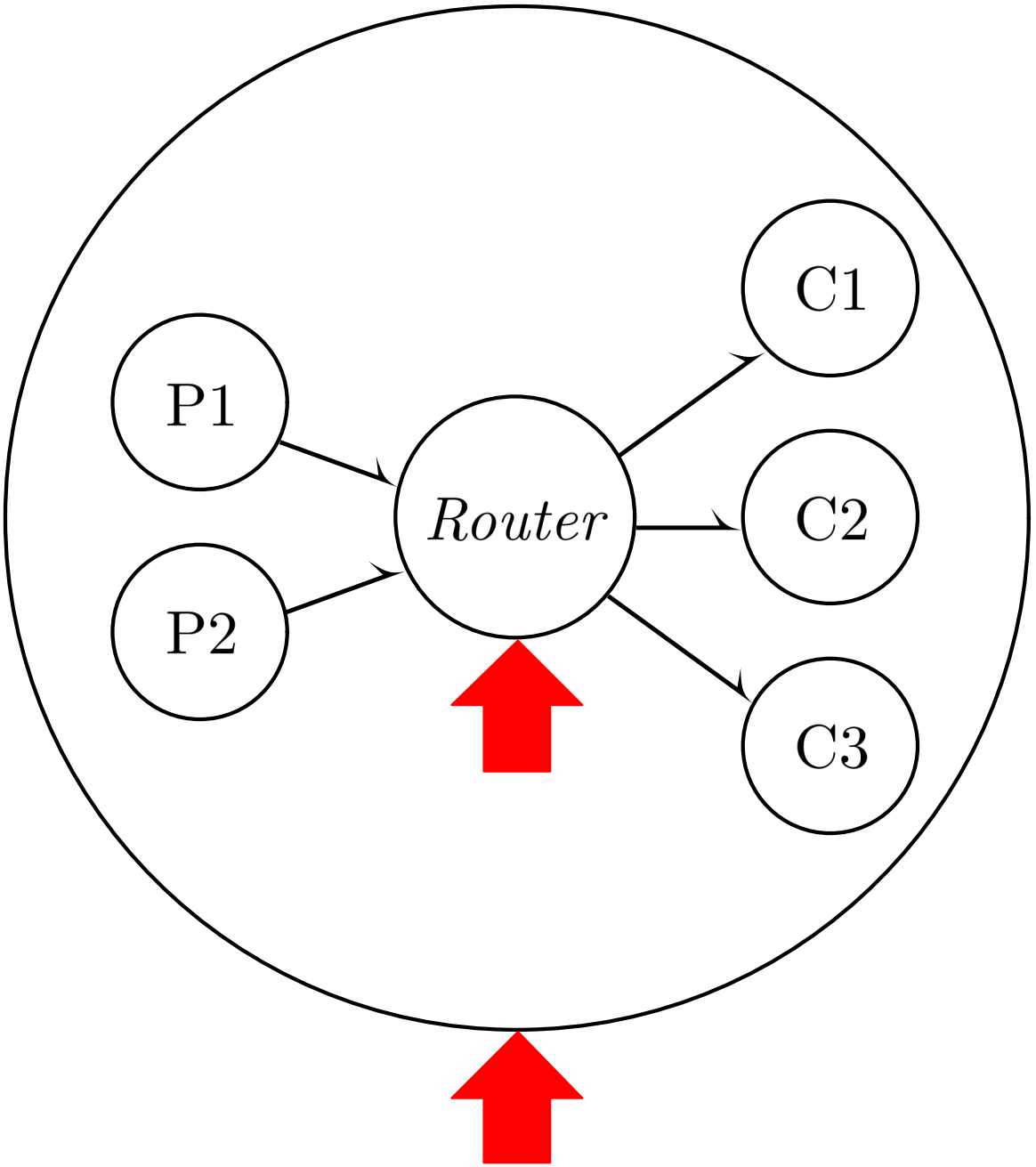
Route log entries from producers to consumers	☐
Consumers filter log messages	☐
Decouple producers from consumers	☐
Avoid shared-state explosion	☐
Discovery of logging service	☐
Only produce if someone's listening	☐
Alert when a producer crashes/exits	☐
Uniform treatment of I/O	☐

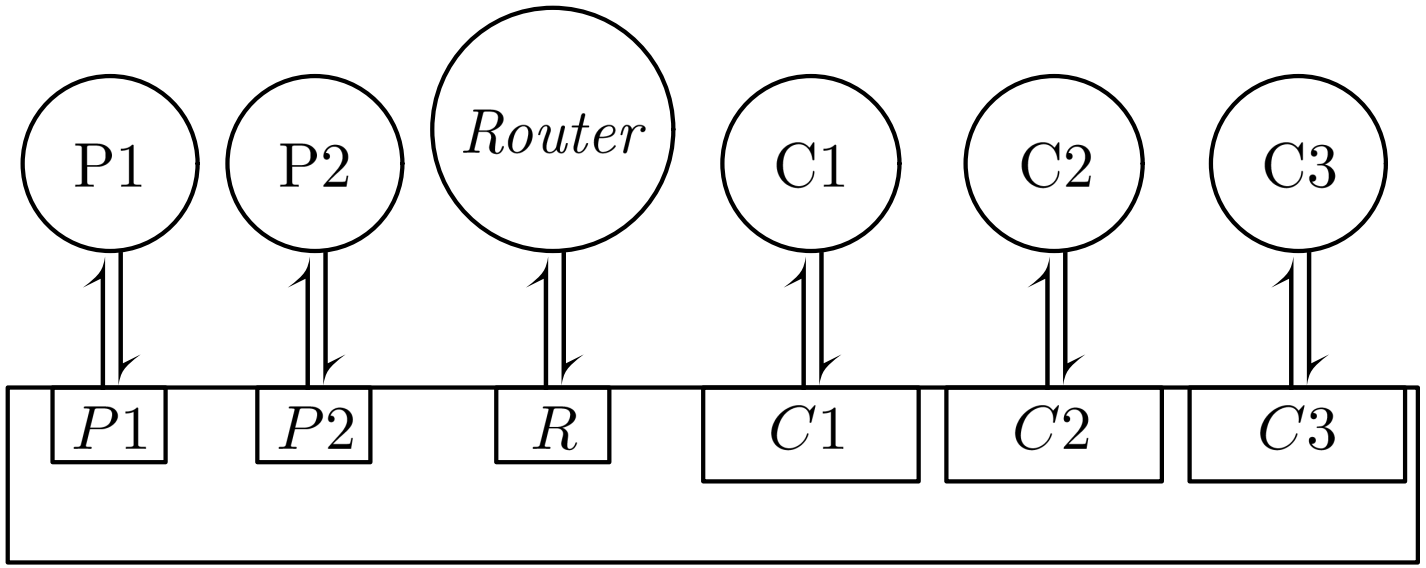
PART II: Why Publish/Subscribe? How?

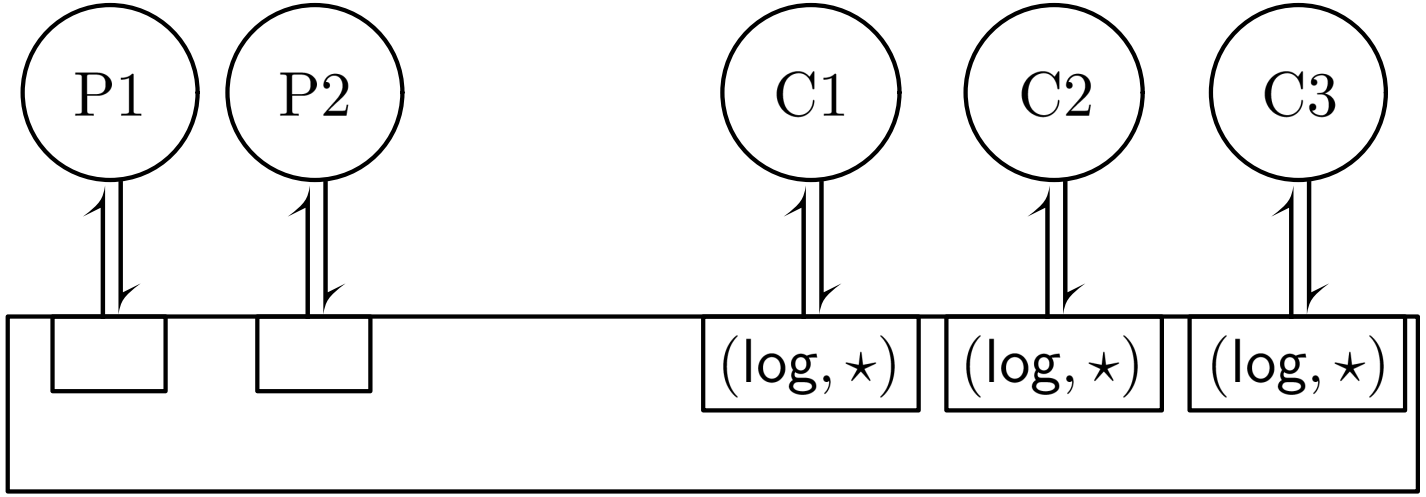

P1
P2
Router
C1
C2
C3

Logging: Requirements Scorecard

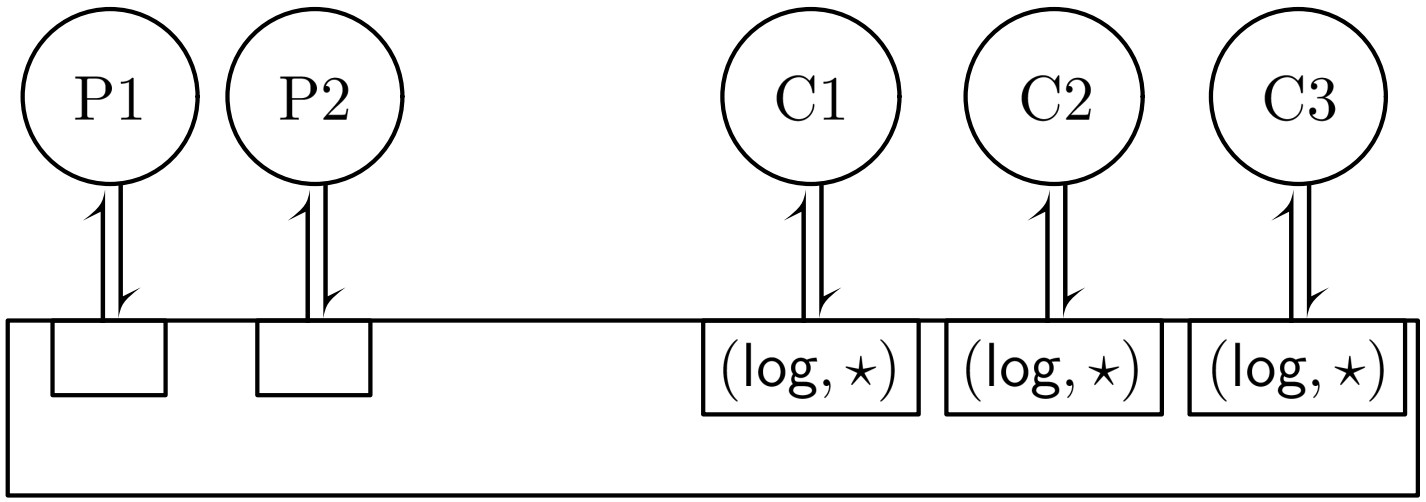
Route log entries from producers to consumers	☑ "Router" actor
Consumers filter log messages	☑ "Router" actor
Decouple producers from consumers	☑ "Router" actor
Avoid shared-state explosion	☑ "Router" actor
Discovery of logging service	☐
Only produce if someone's listening	☐
Alert when a producer crashes/exits	☐
Uniform treatment of I/O	☐


P1
P2
Router
C1
C2
C3

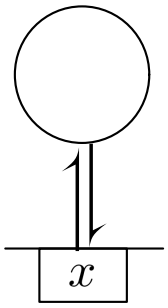

P1
P2
Router
C1
C2
C3
P1
P2
R
C1
C2
C3



See Eugster's 2003 pub/sub survey

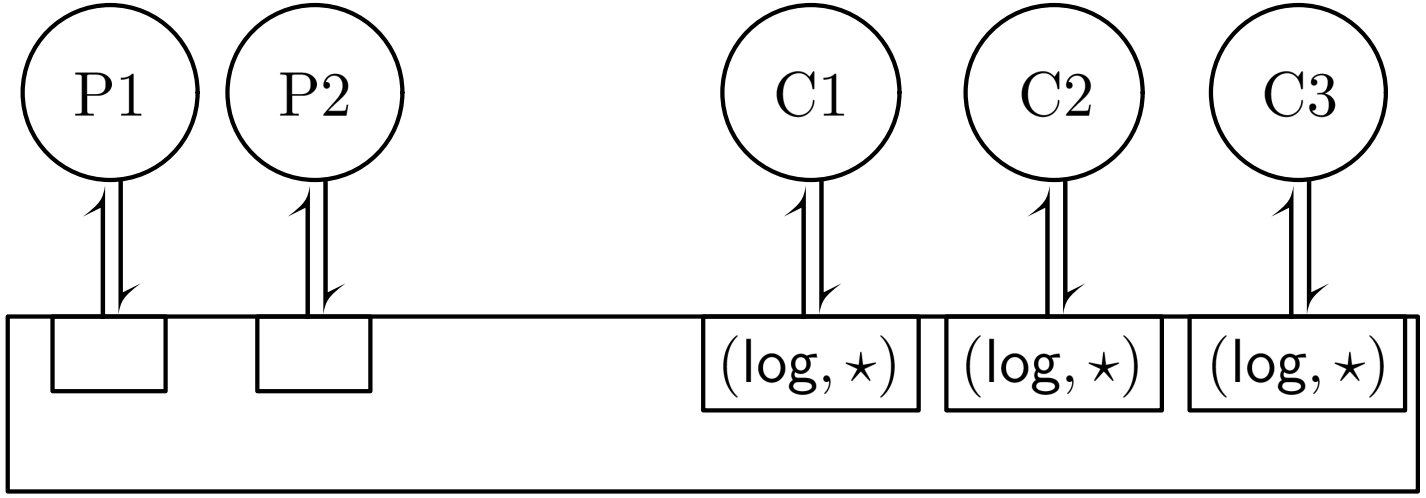


Route by address

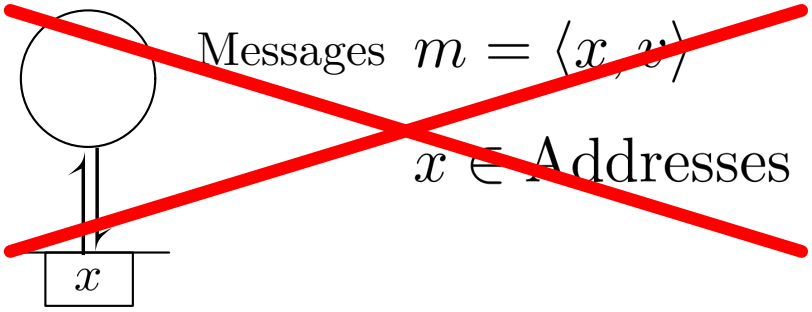


Messages $m = \langle x, v \rangle$

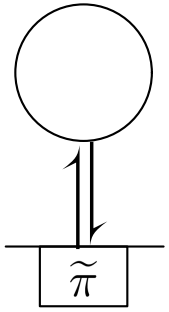
$x \in$ Addresses



Route by address



Route by content

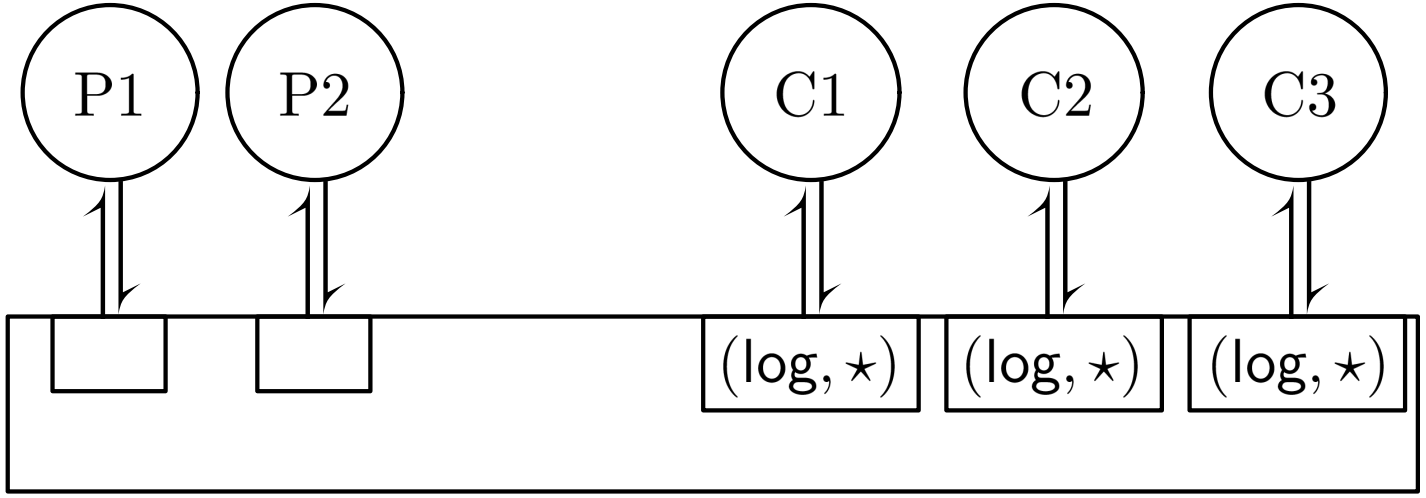


$$m = \langle v \rangle$$

$$v = u \mid v, v$$

Patterns $$p = u \mid p, p \mid \star$$

Interests $$\pi = (p)$$



Route by address

$(C1, \star)$

Route by content

$(\mathsf{log}, \star)$

or

$(\mathsf{log}, [\star, \mathsf{error}, \star])$

or

$(\mathsf{log}, [\mathrm{P1}, \star, \star])$

or

...

Logging: Requirements Scorecard

Route log entries from producers to consumers	☑ pub/sub
Consumers filter log messages	☑ pub/sub
Decouple producers from consumers	☑ pub/sub
Avoid shared-state explosion	☑ pub/sub
Discovery of logging service	⊟ no need!
Only produce if someone's listening	☐
Alert when a producer crashes/exits	☐
Uniform treatment of I/O	☑ pub/sub

PART III: Why Routing Events? How?

Logging: Requirements Scorecard

Route log entries from producers to consumers	☑
Consumers filter log messages	☑
Decouple producers from consumers	☑
Avoid shared-state explosion	☑
Discovery of logging service	☐
Only produce if someone's listening	☐
Alert when a producer crashes/exits	☐
Uniform treatment of I/O	☑

Shared Conversational Interest

Shared Conversational Interest

$$\underset{\text{Interests}}{\pi} \quad = \quad \underset{\text{Subscription}}{(p)}$$

Shared Conversational Interest

$$\underset{\text{Interests}}{\pi} = \underset{\text{Subscription}}{(p)} \mid \underset{\text{Advertisement}}{\langle p \rangle}$$

Shared Conversational Interest

Interests: π = Subscription: (p) | Advertisement: $\langle p \rangle$

$$\langle p \rangle \curlywedge \langle p' \rangle = \emptyset$$
$$(q) \curlywedge (q') = \emptyset$$

$$\langle p \rangle \curlywedge (q) = \langle p \cap q \rangle$$
$$(q) \curlywedge \langle p \rangle = (p \cap q)$$

Shared Conversational Interest

$$\underset{\text{Interests}}{\pi} = \underset{\text{Subscription}}{(p)} \mid \underset{\text{Advertisement}}{\langle p \rangle}$$

$$\langle p \rangle \pitchfork \langle p' \rangle = \emptyset$$
$$(q) \pitchfork (q') = \emptyset$$

$$\langle p \rangle \pitchfork (q) = \langle p \cap q \rangle$$
$$(q) \pitchfork \langle p \rangle = (p \cap q)$$

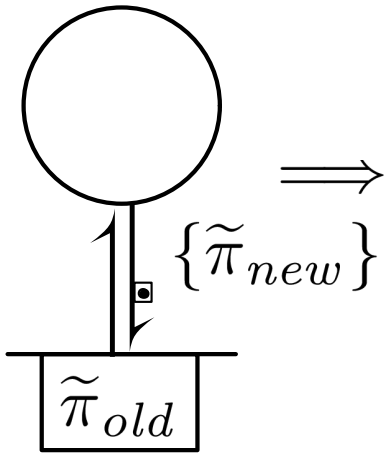
Any pattern language will do — if it supports $\cap$

What is a Routing Event?

$$\{\widetilde{\pi}\}$$

What is a Routing Event?

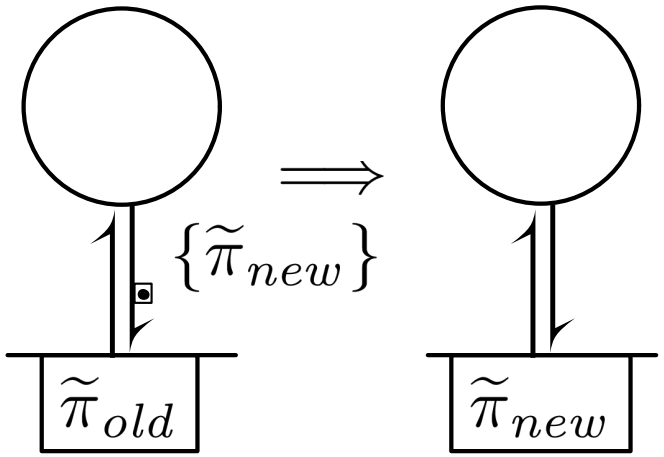
$$\{\widetilde{\pi}\}$$



From Actor to Network

What is a Routing Event?

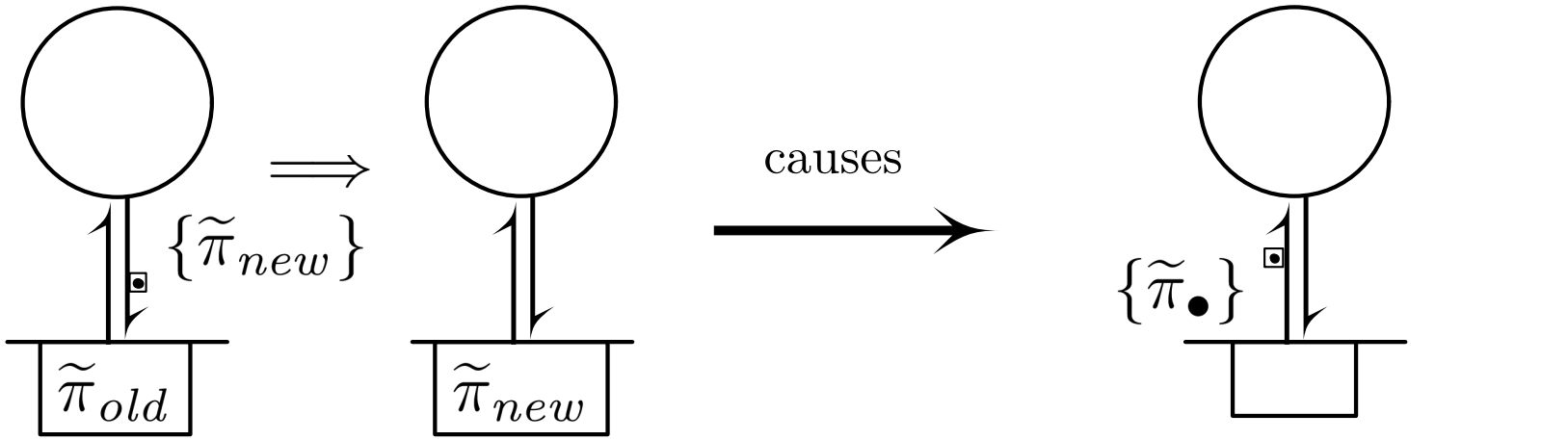
$$\{\widetilde{\pi}\}$$



From Actor to Network

What is a Routing Event?

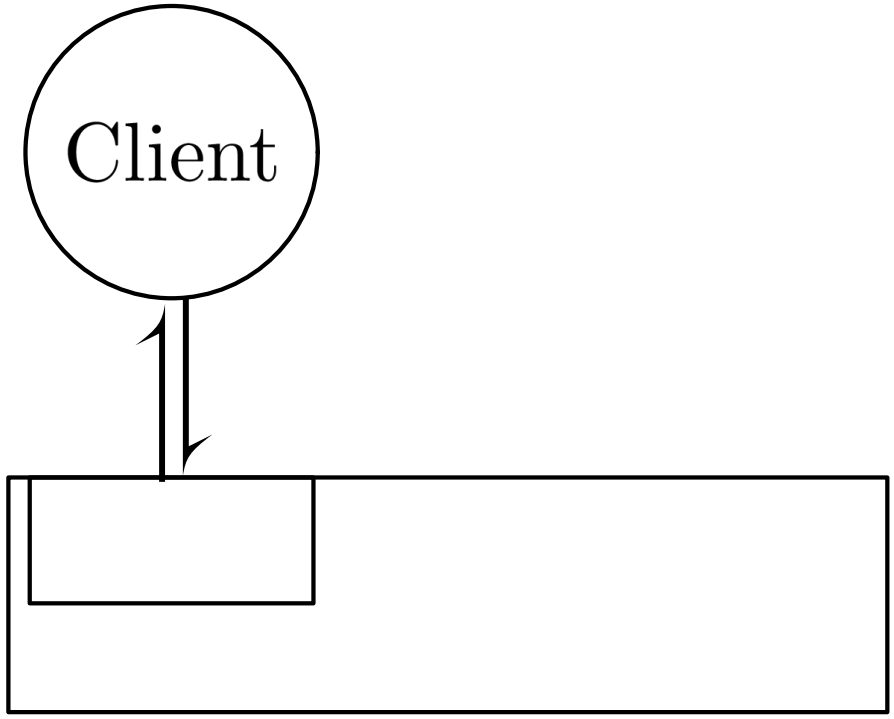
$\{\widetilde{\pi}\}$



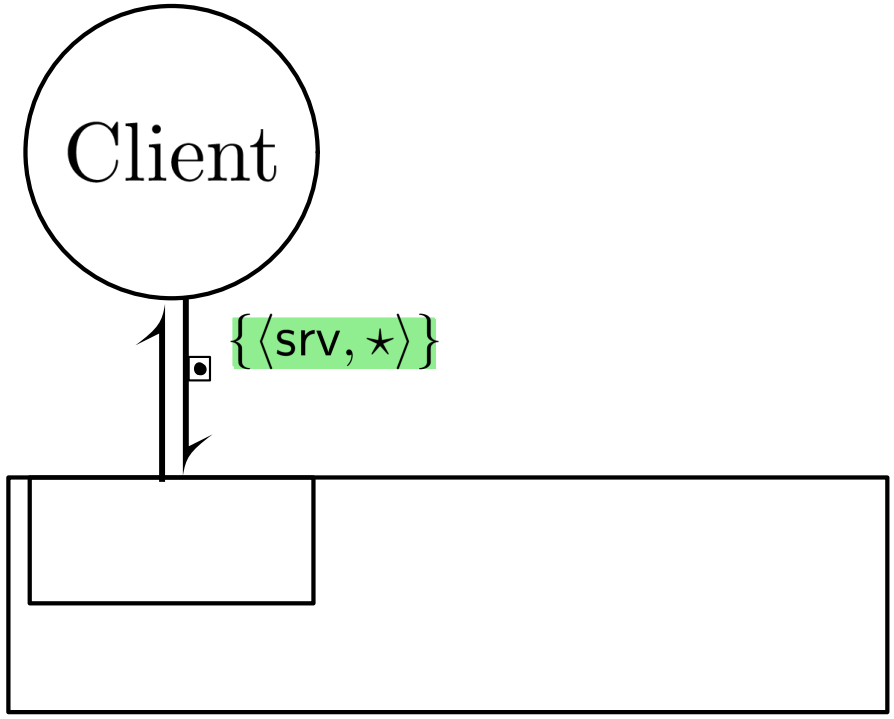
From Actor to Network

From Network to Actor

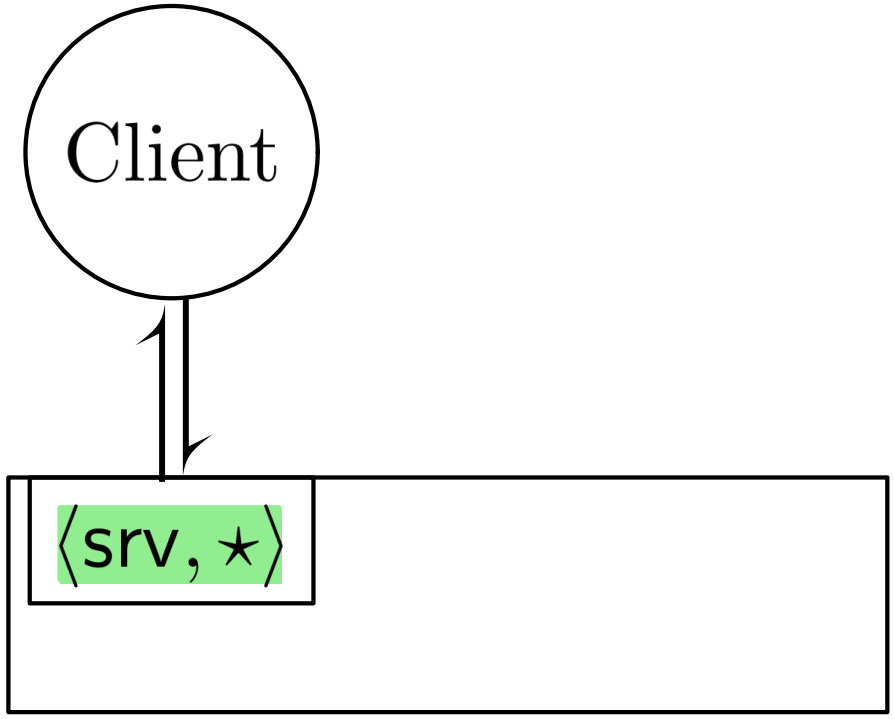
Routing Events for Service Discovery



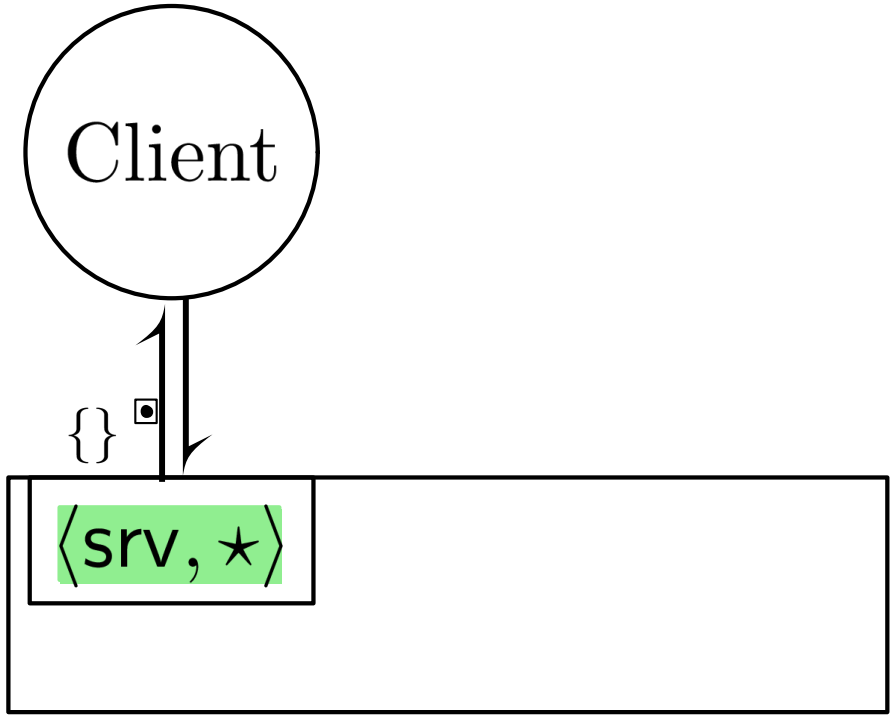
Routing Events for Service Discovery



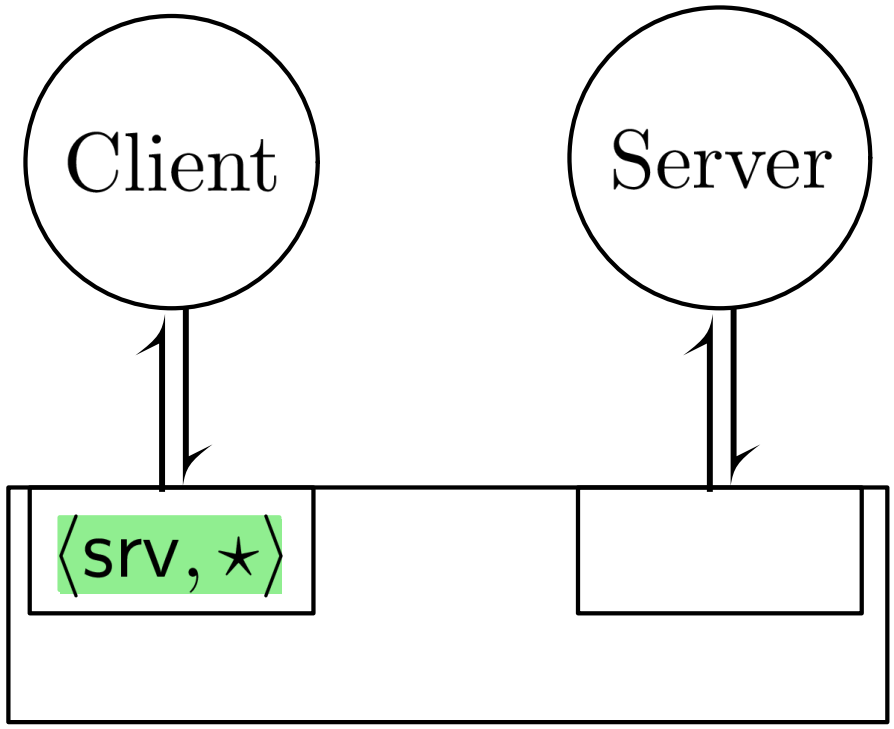
Routing Events for Service Discovery



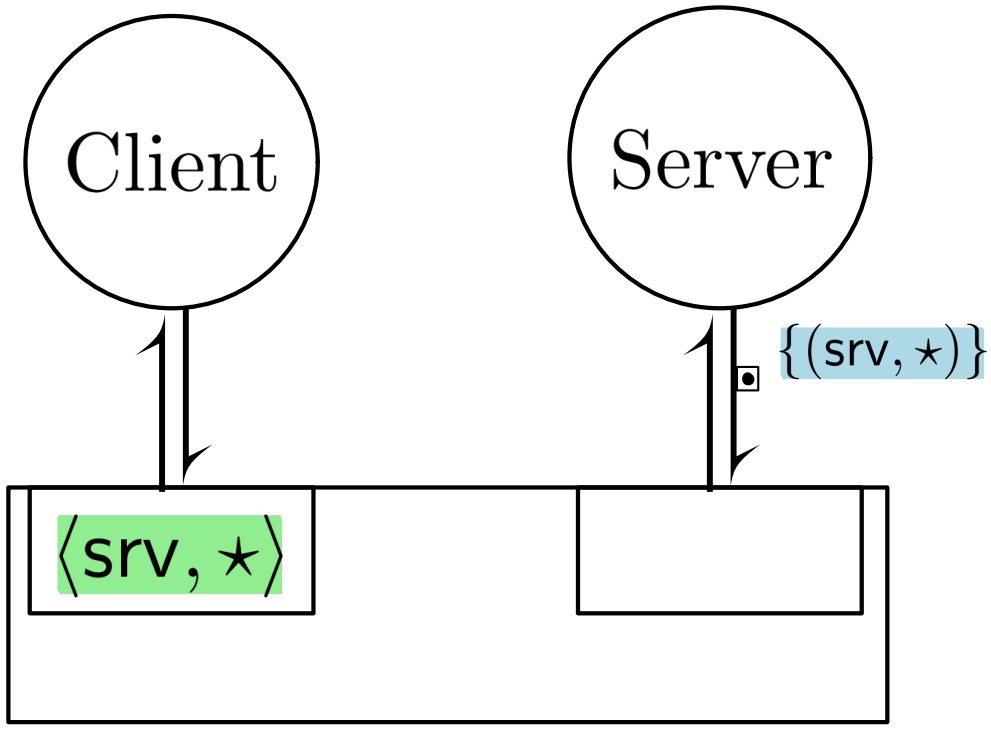
Routing Events for Service Discovery



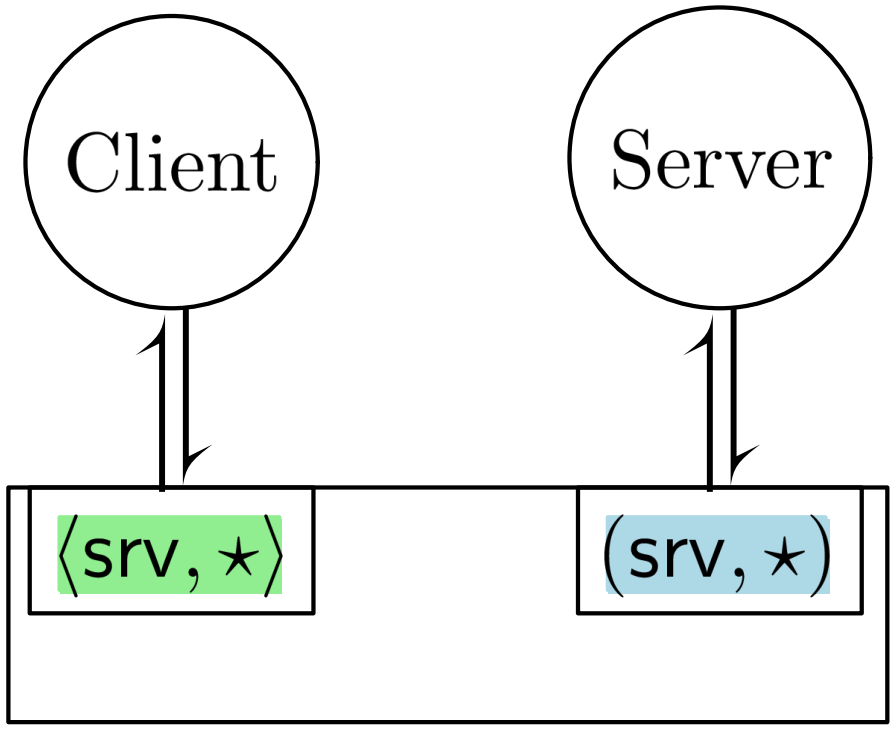
Routing Events for Service Discovery



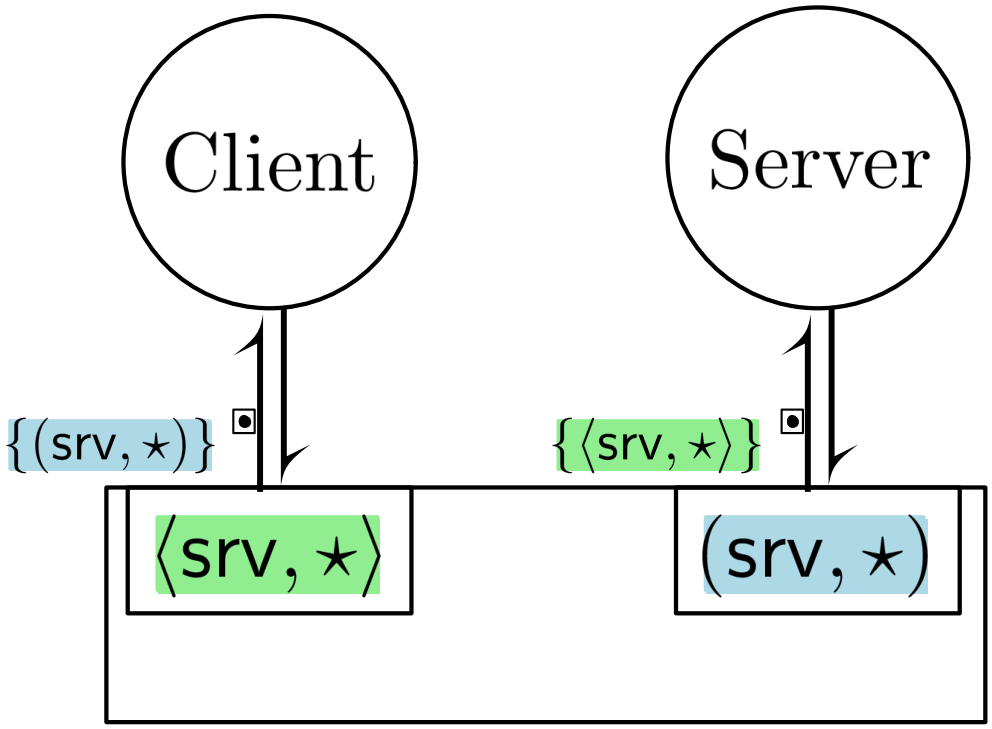
Routing Events for Service Discovery



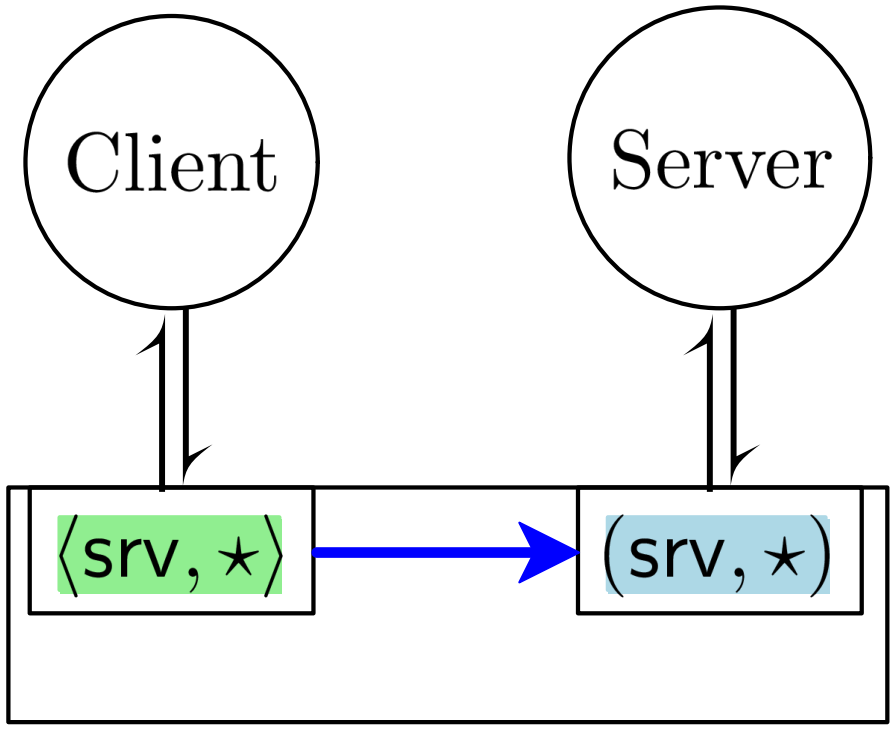
Routing Events for Service Discovery



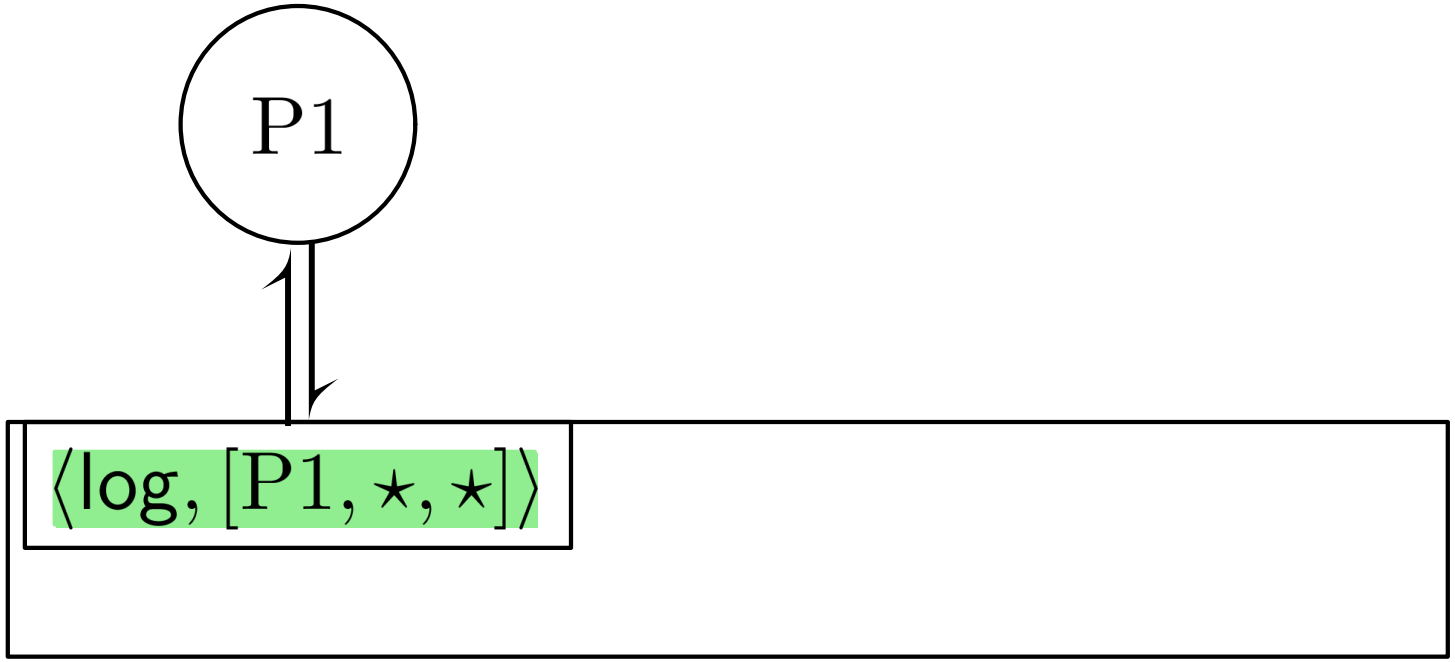
Routing Events for Service Discovery



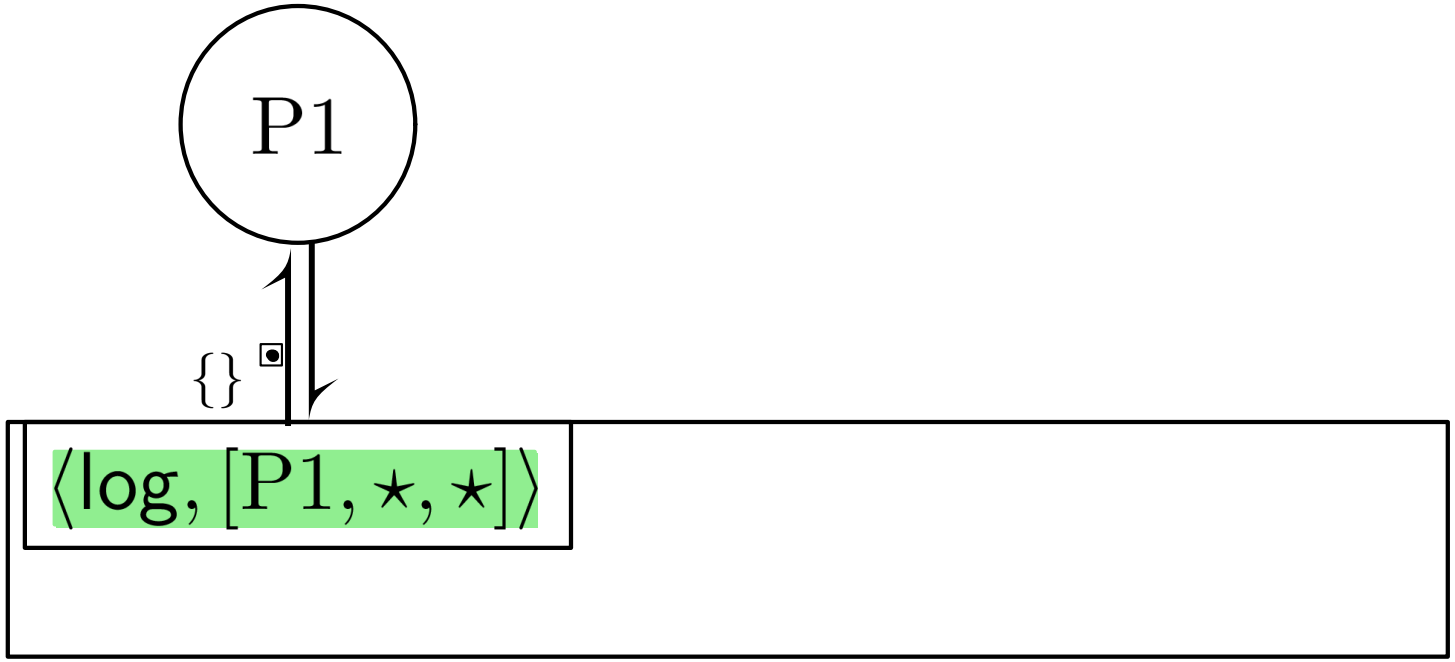
Routing Events for Service Discovery



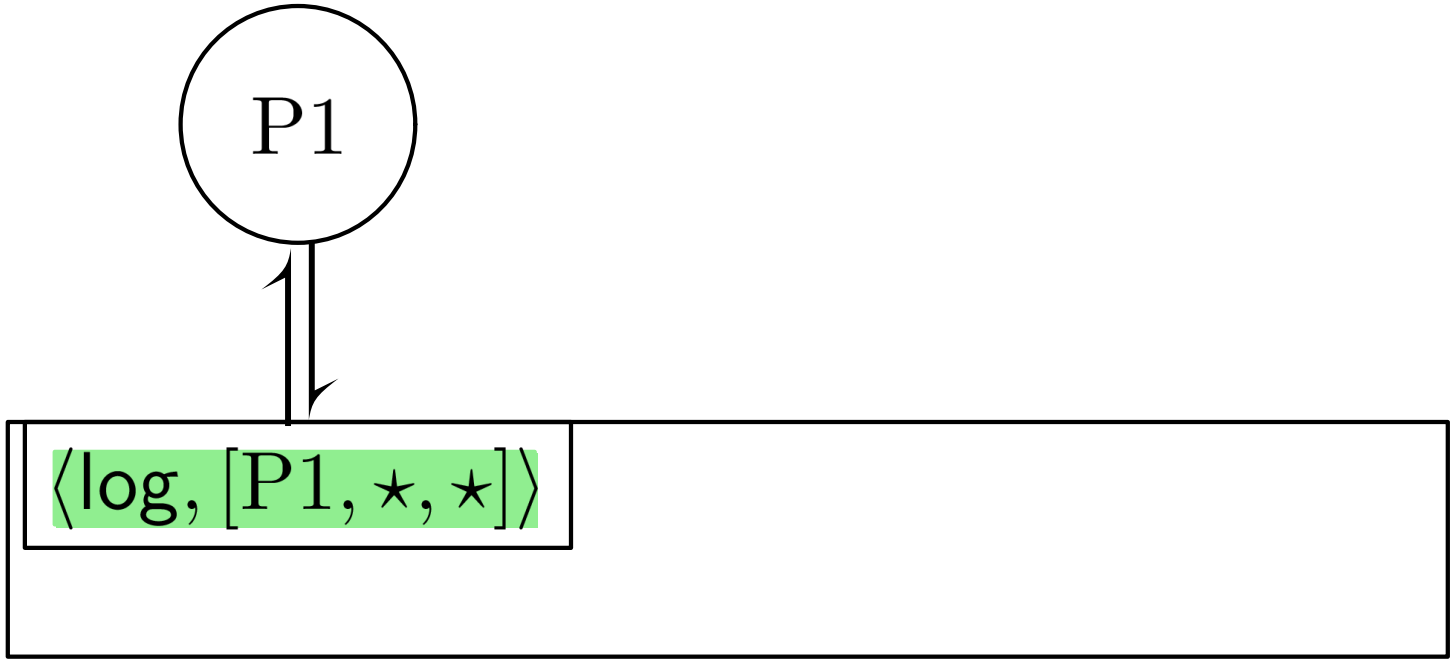
Routing Events for Presence Detection



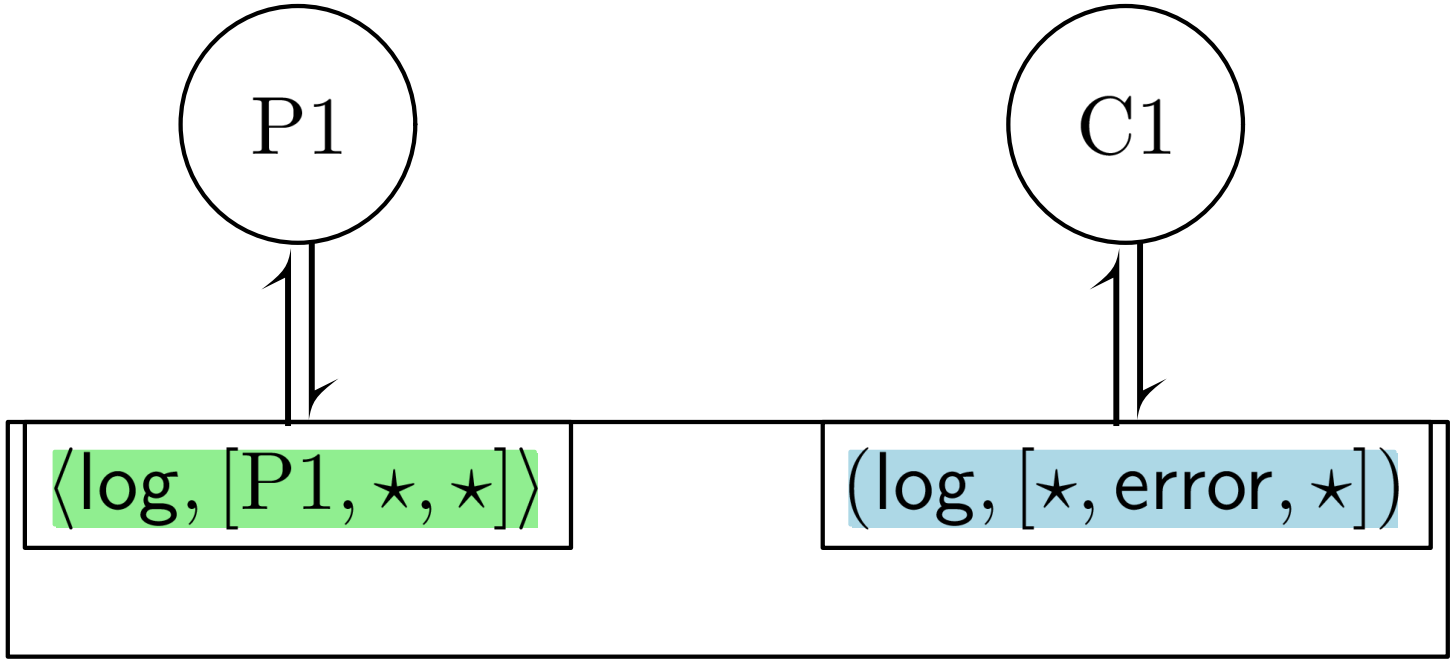
Routing Events for Presence Detection



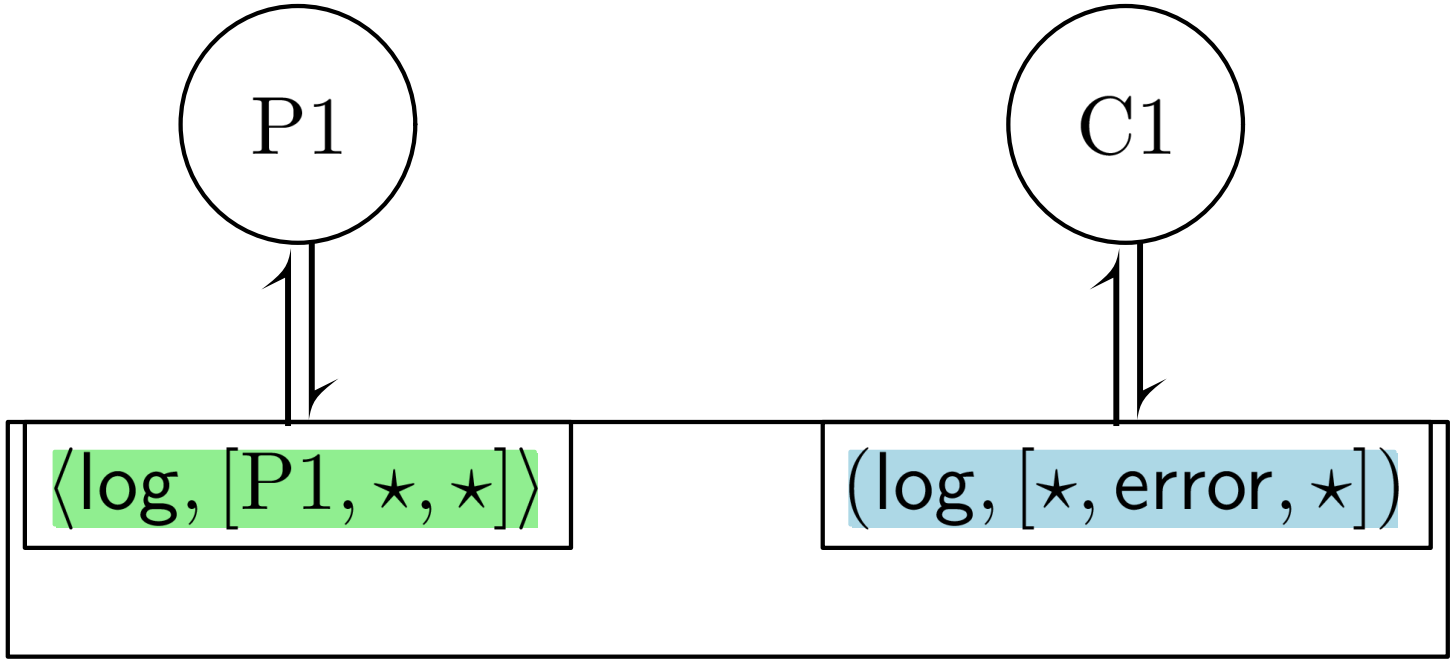
Routing Events for Presence Detection



Routing Events for Presence Detection

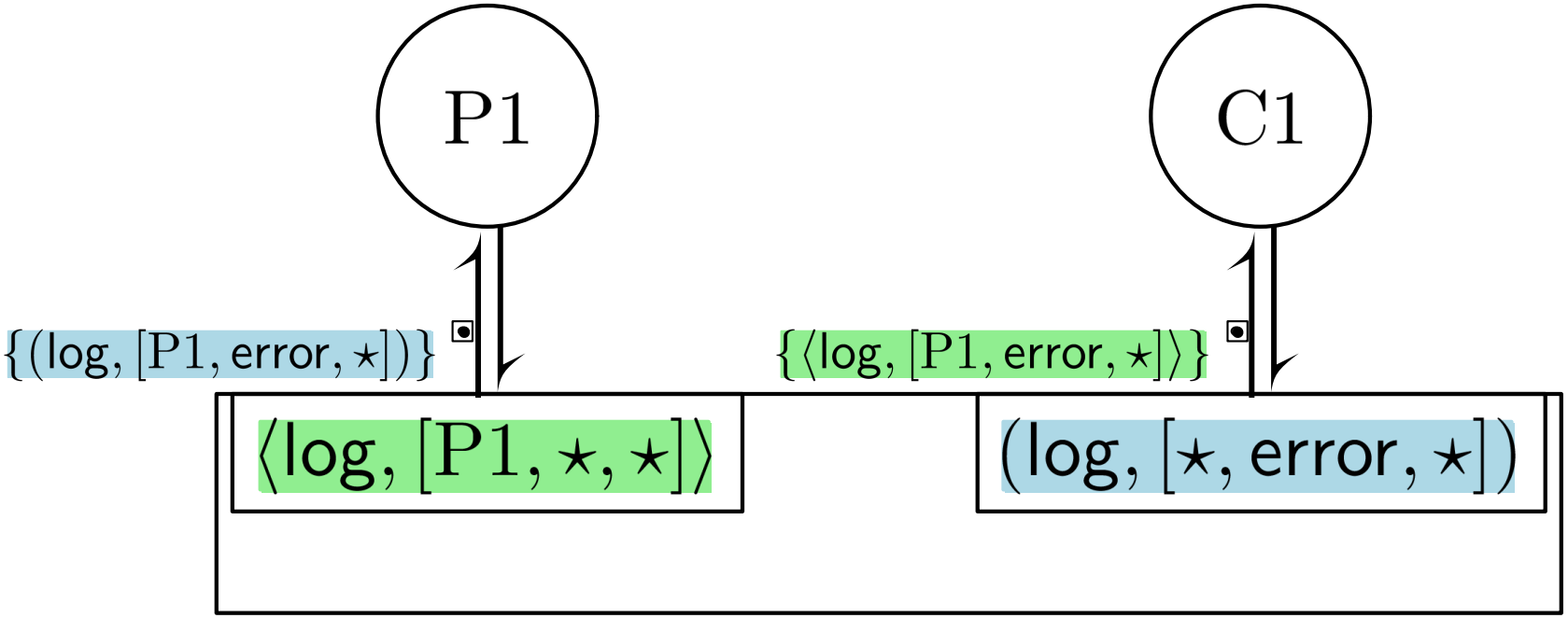


Routing Events for Presence Detection



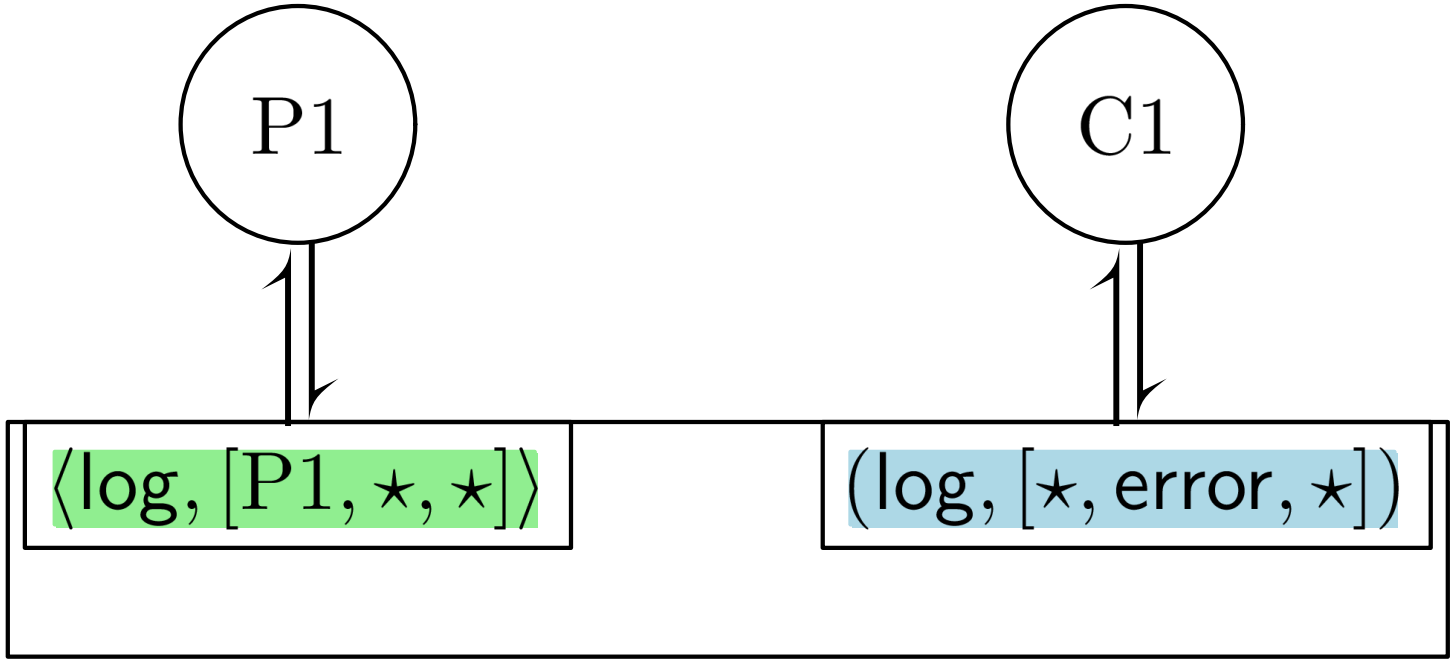
$$\mathsf{log}, [\mathrm{P1}, \star, \star] \quad \cap \quad \mathsf{log}, [\star, \mathsf{error}, \star] \quad = \quad \mathsf{log}, [\mathrm{P1}, \mathsf{error}, \star]$$

Routing Events for Presence Detection



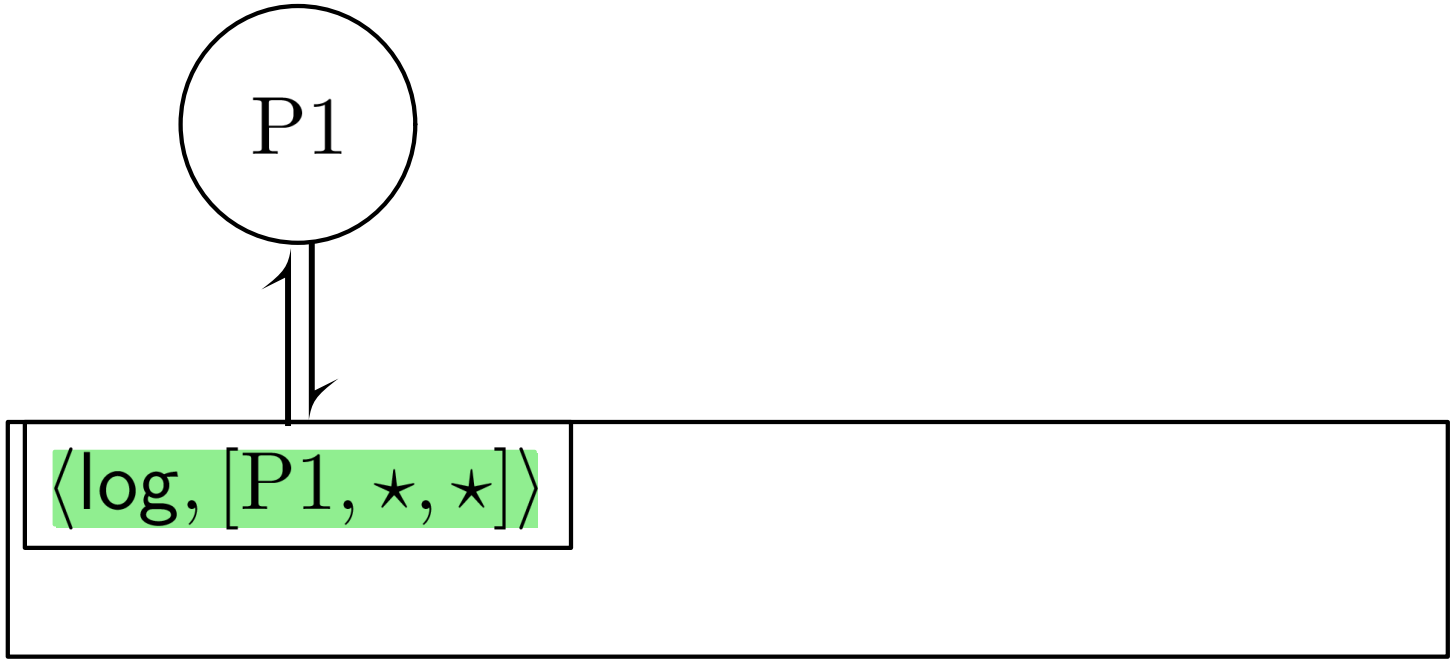
$$\text{log}, [\text{P1}, \star, \star] \quad \cap \quad \text{log}, [\star, \text{error}, \star] \quad = \quad \text{log}, [\text{P1}, \text{error}, \star]$$

Routing Events for Presence Detection

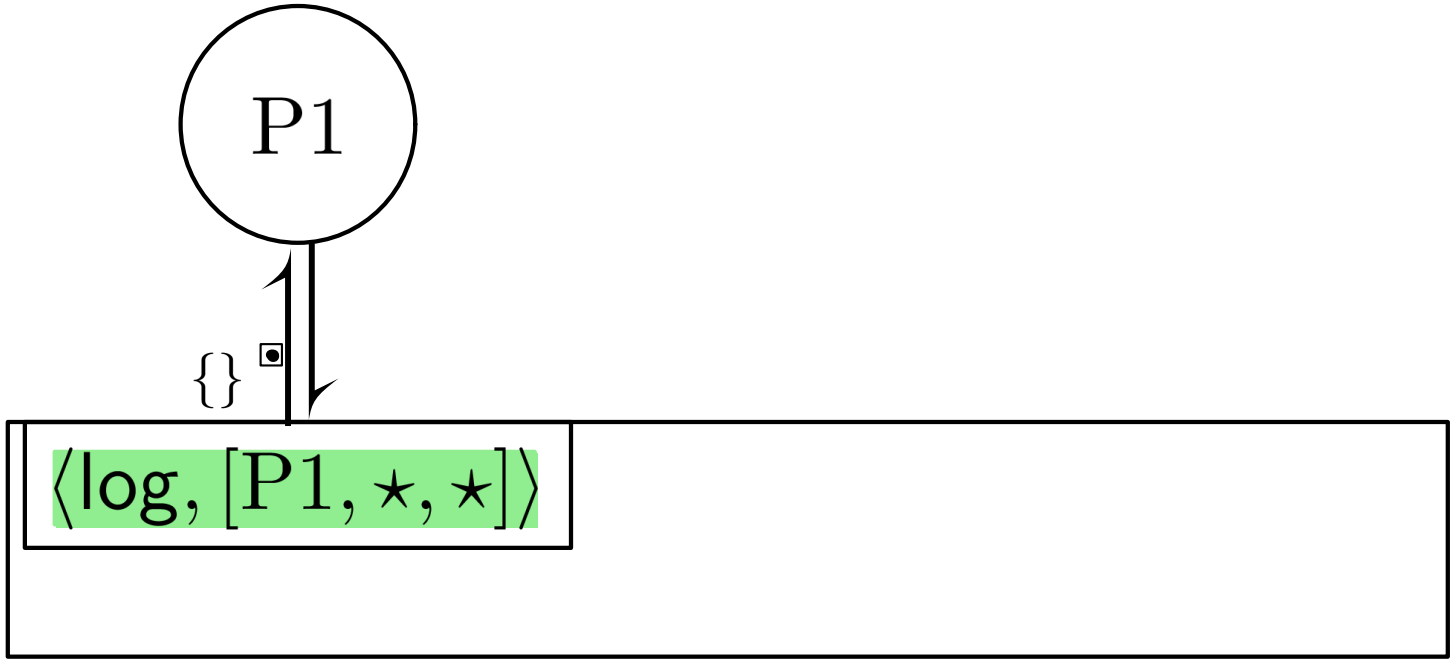


$$\mathsf{log}, [\mathrm{P1}, \star, \star] \quad \cap \quad \mathsf{log}, [\star, \mathsf{error}, \star] \quad = \quad \mathsf{log}, [\mathrm{P1}, \mathsf{error}, \star]$$

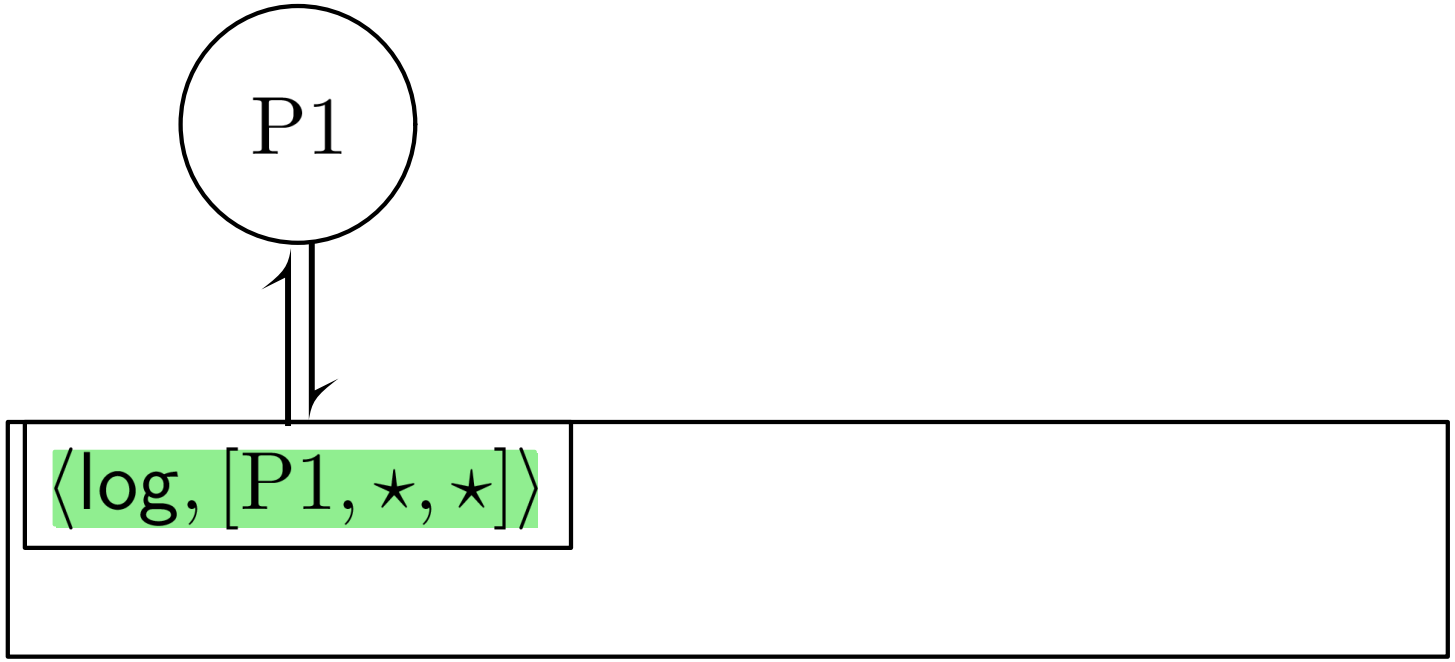
Routing Events for Presence Detection



Routing Events for Presence Detection



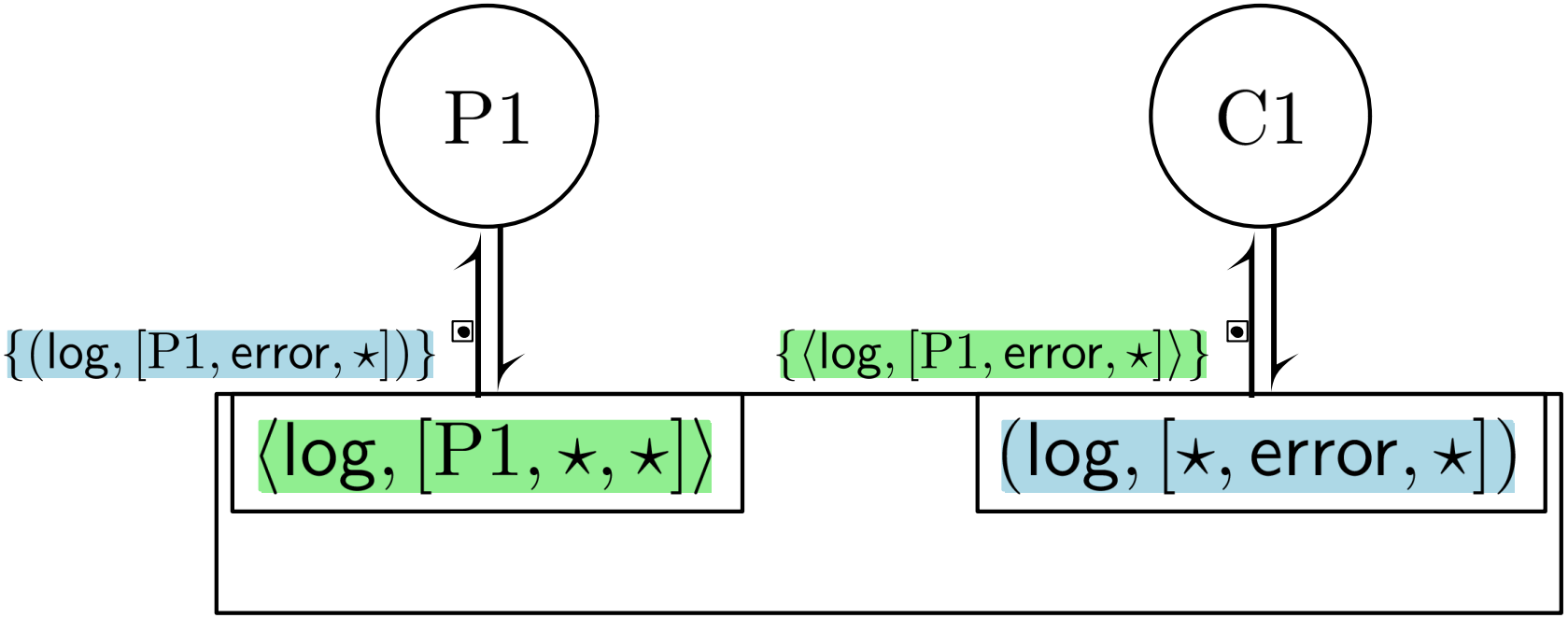
Routing Events for Presence Detection



Routing Events for Crash Detection

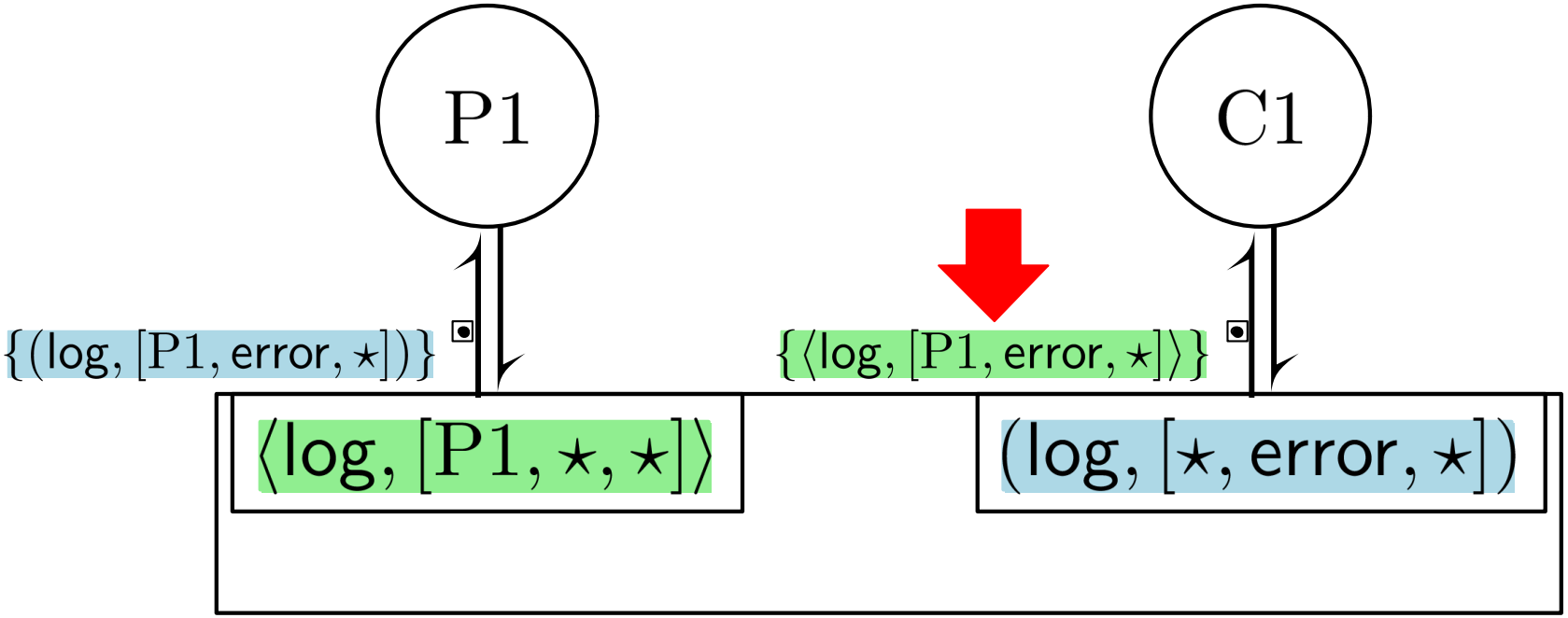
cf. Erlang's links/monitors [Armstrong 2003]

Routing Events for Crash Detection



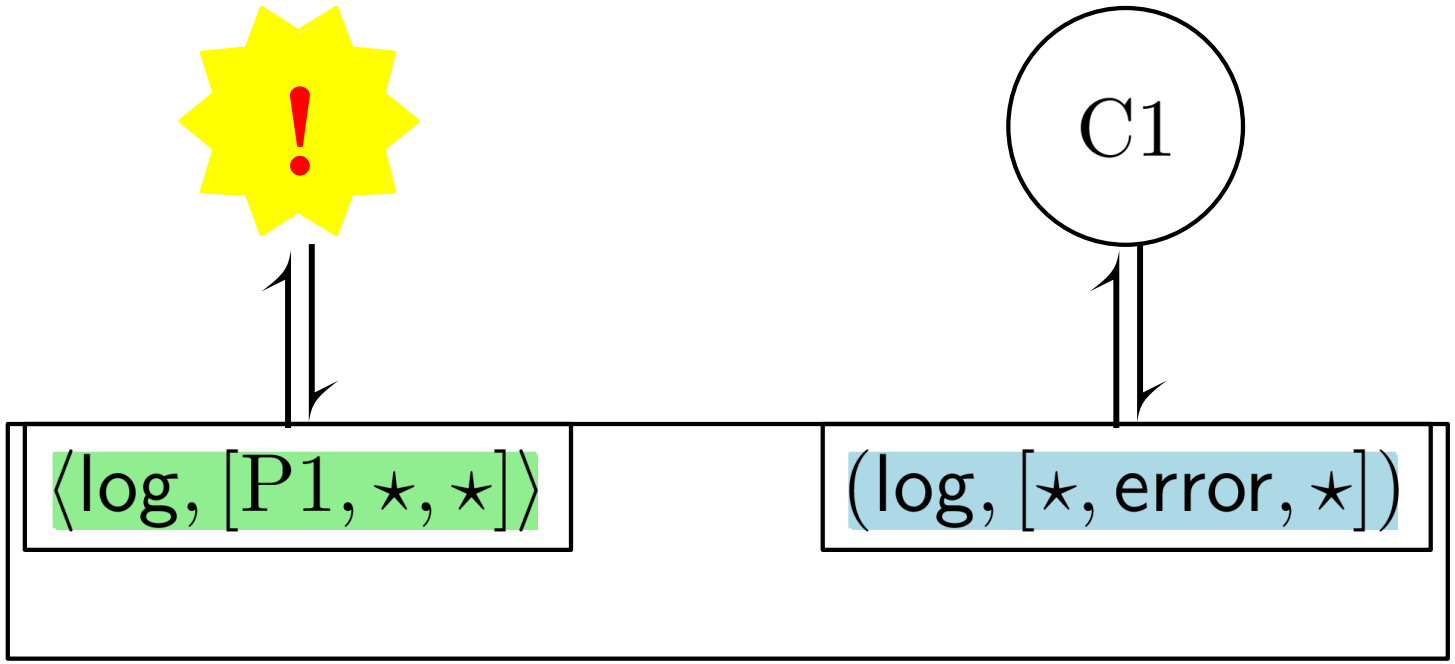
cf. Erlang's links/monitors [Armstrong 2003]

Routing Events for Crash Detection



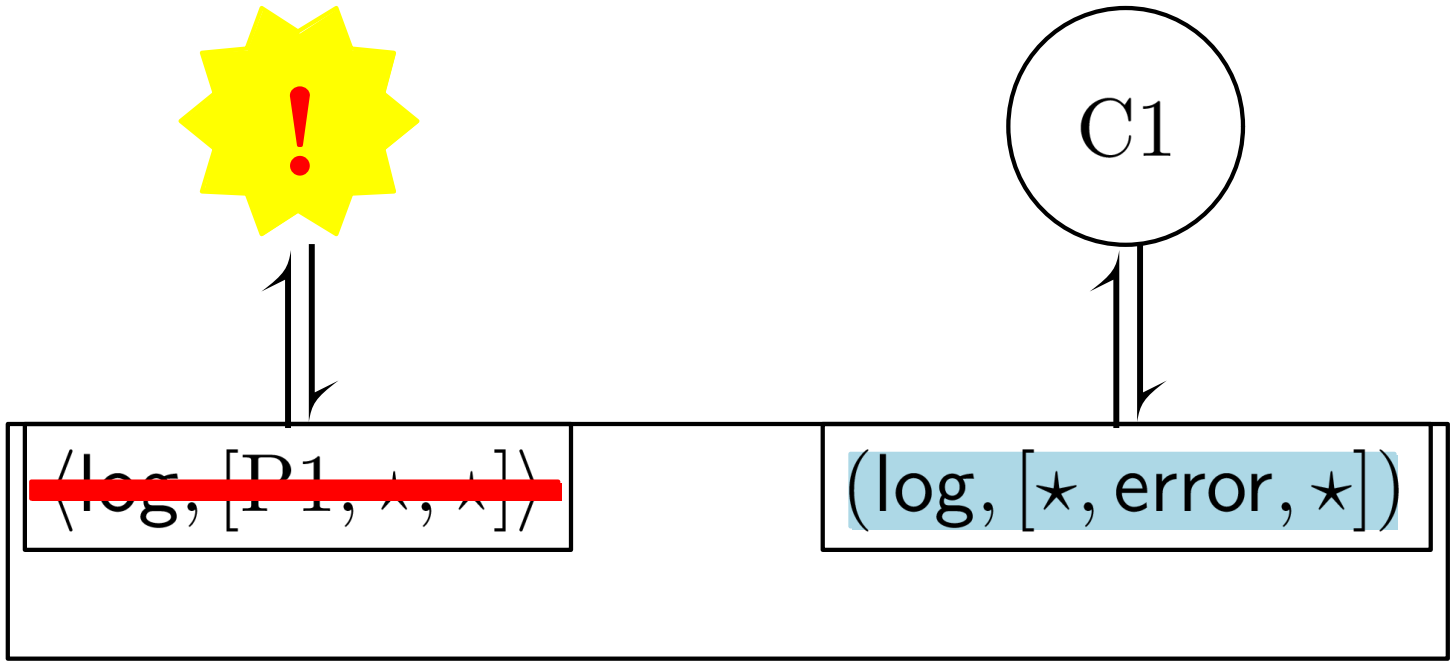
cf. Erlang's links/monitors [Armstrong 2003]

Routing Events for Crash Detection



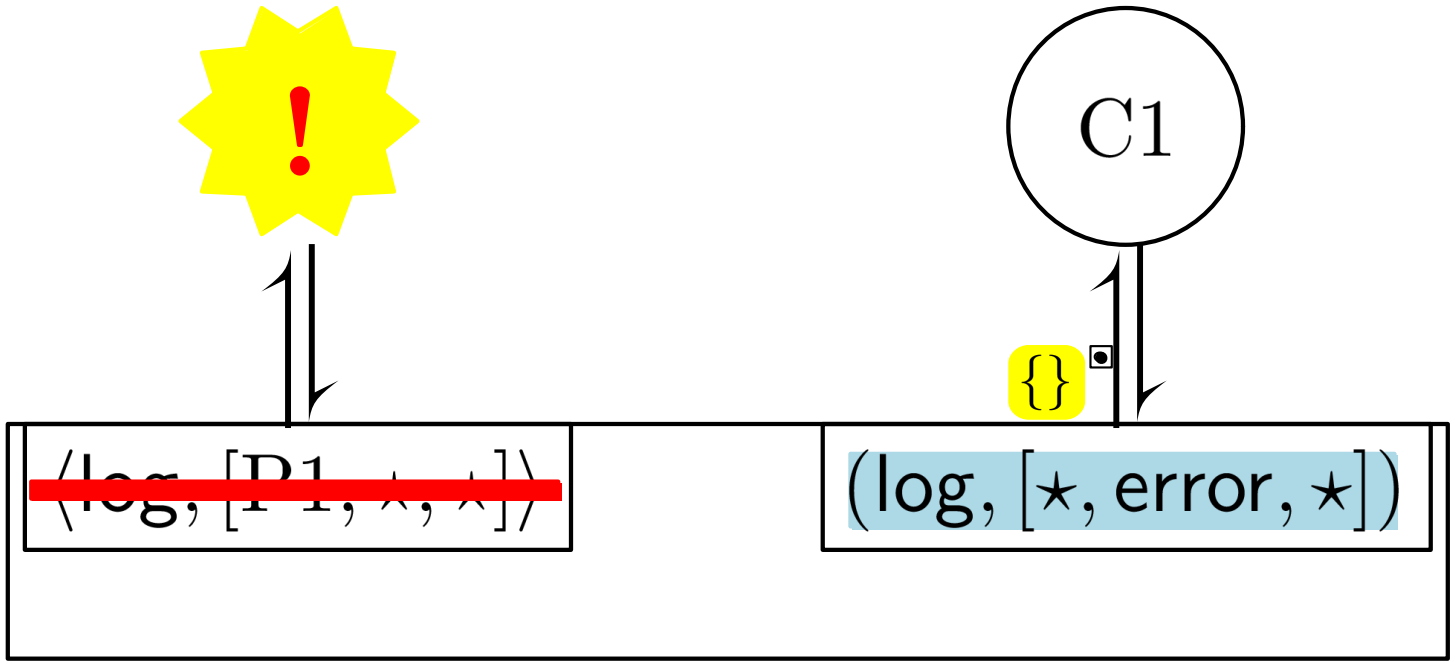
cf. Erlang's links/monitors [Armstrong 2003]

Routing Events for Crash Detection



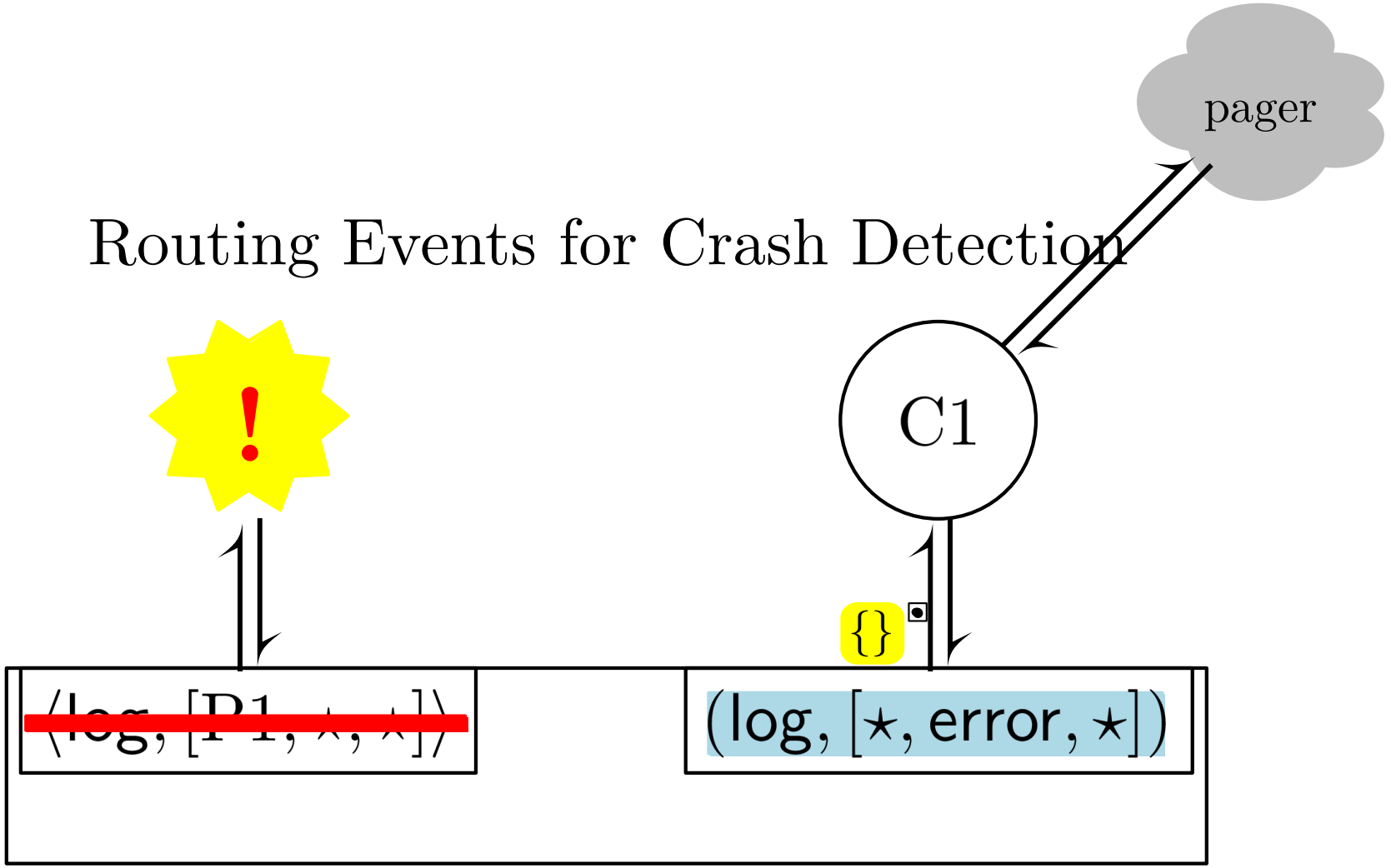
cf. Erlang's links/monitors [Armstrong 2003]

Routing Events for Crash Detection



cf. Erlang's links/monitors [Armstrong 2003]

Routing Events for Crash Detection



cf. Erlang's links/monitors [Armstrong 2003]

Logging: Requirements Scorecard

Route log entries from producers to consumers	☑ pub/sub
Consumers filter log messages	☑ pub/sub
Decouple producers from consumers	☑ pub/sub
Avoid shared-state explosion	☑ pub/sub
Discovery of logging service	☑ routing events
Only produce if someone's listening	☑ routing events
Alert when a producer crashes/exits	☑ routing events
Uniform treatment of I/O	☑ pub/sub

Logging: Requirements Scorecard

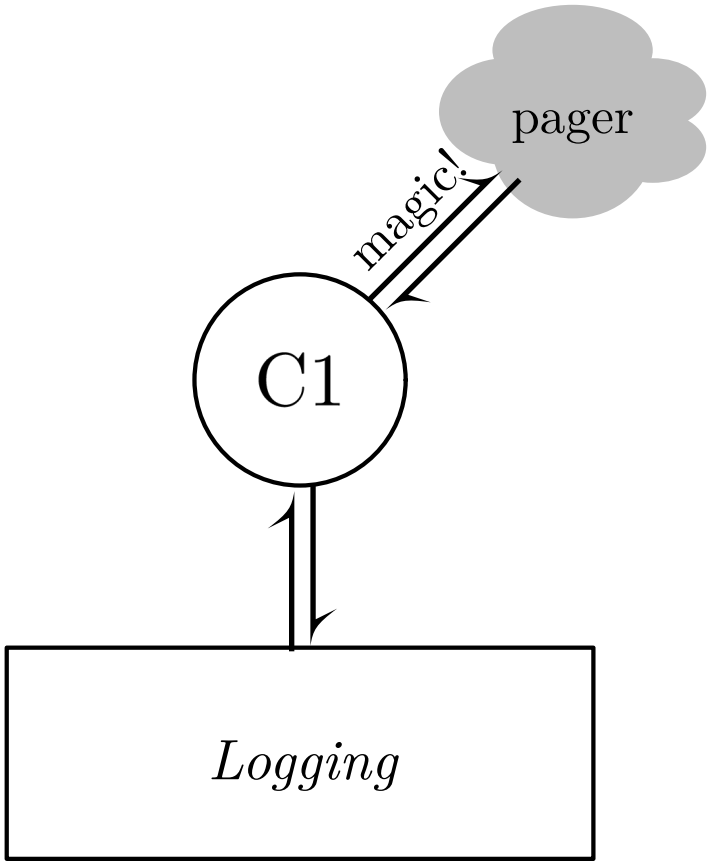
Route log entries from producers to consumers	☑ pub/sub
Consumers filter log messages	☑ pub/sub
Decouple producers from consumers	☑ pub/sub
Avoid shared-state explosion	☑ pub/sub
Discovery of logging service	☑ routing events
Only produce if someone's listening	☑ routing events
Alert when a producer crashes/exits	☑ routing events
Uniform treatment of I/O	⛀ not finished!

PART IV: Why Hierarchical Layering? How?

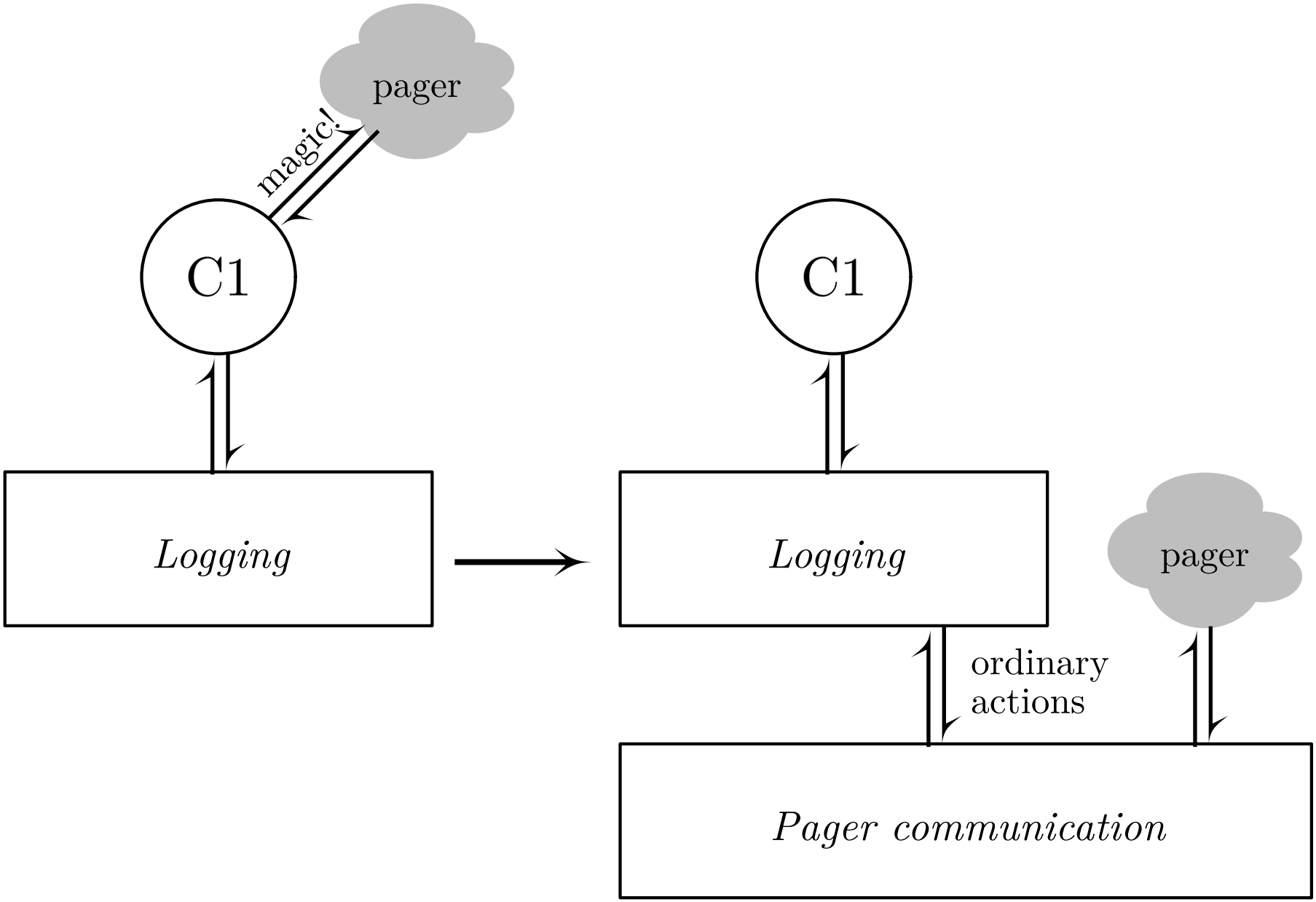
Logging: Requirements Scorecard

Route log entries from producers to consumers	☑
Consumers filter log messages	☑
Decouple producers from consumers	☑
Avoid shared-state explosion	☑
Discovery of logging service	☑
Only produce if someone's listening	☑
Alert when a producer crashes/exits	☑
Uniform treatment of I/O	☐ not finished!

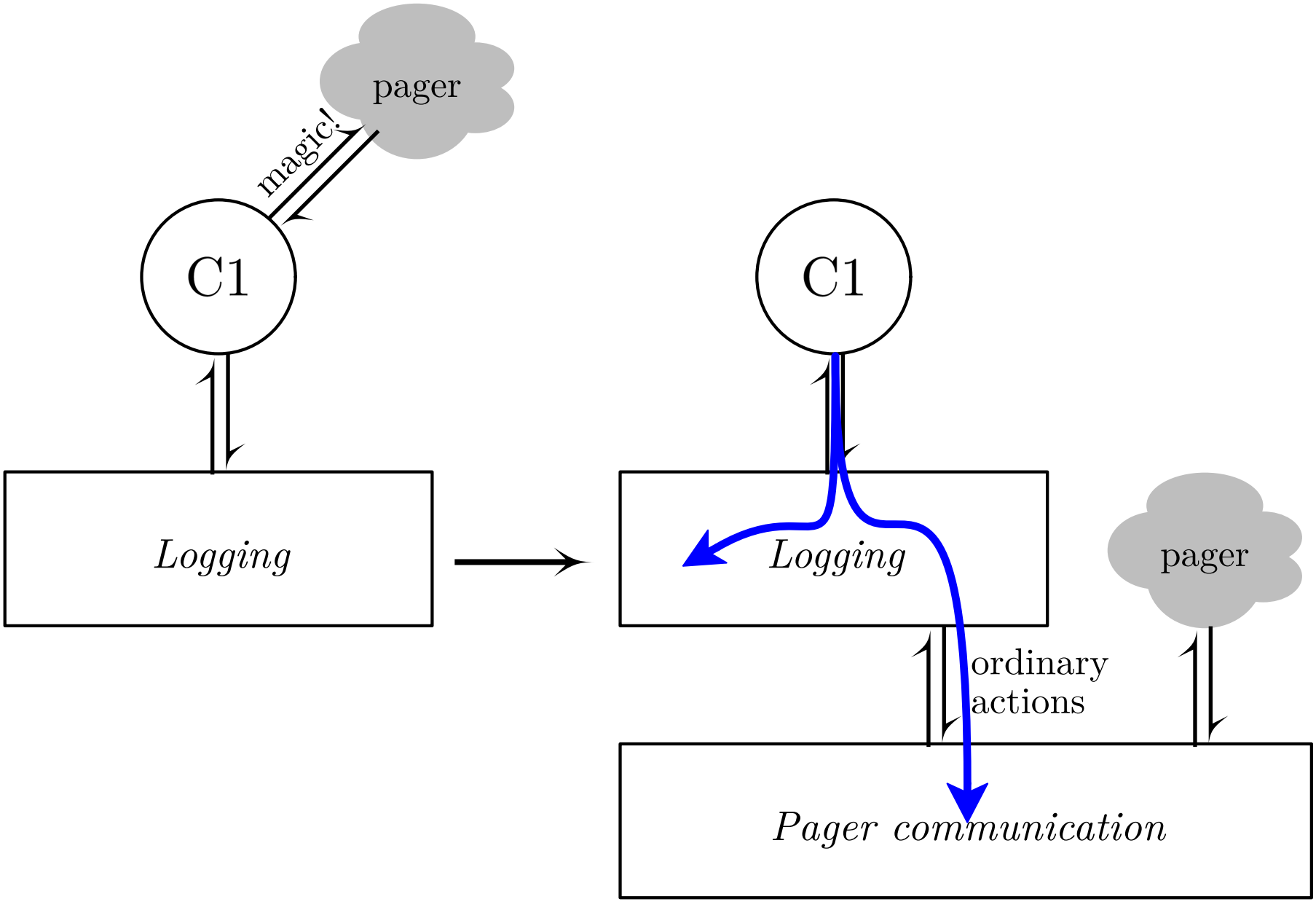
Layers make I/O Uniform



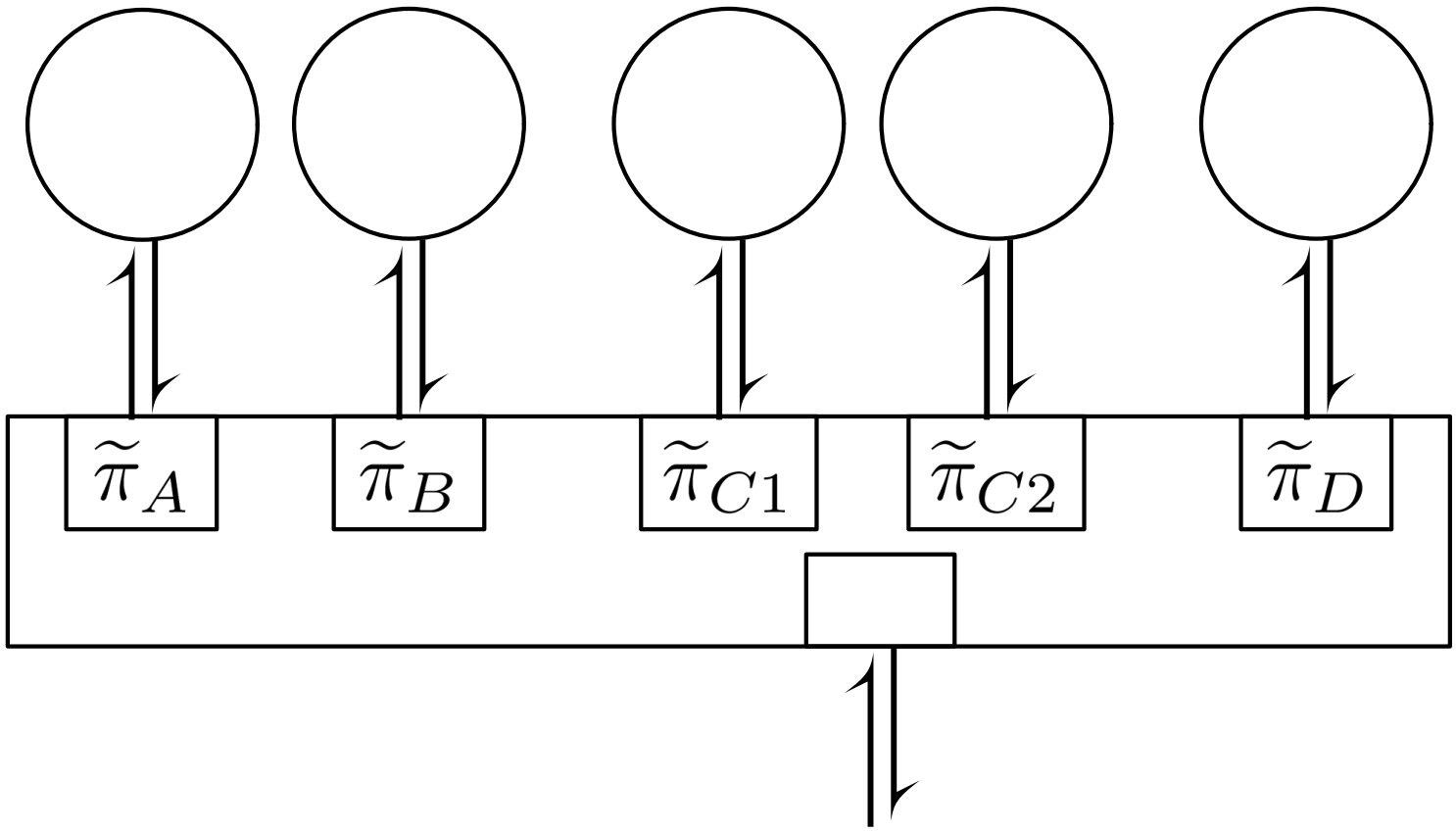
Layers make I/O Uniform



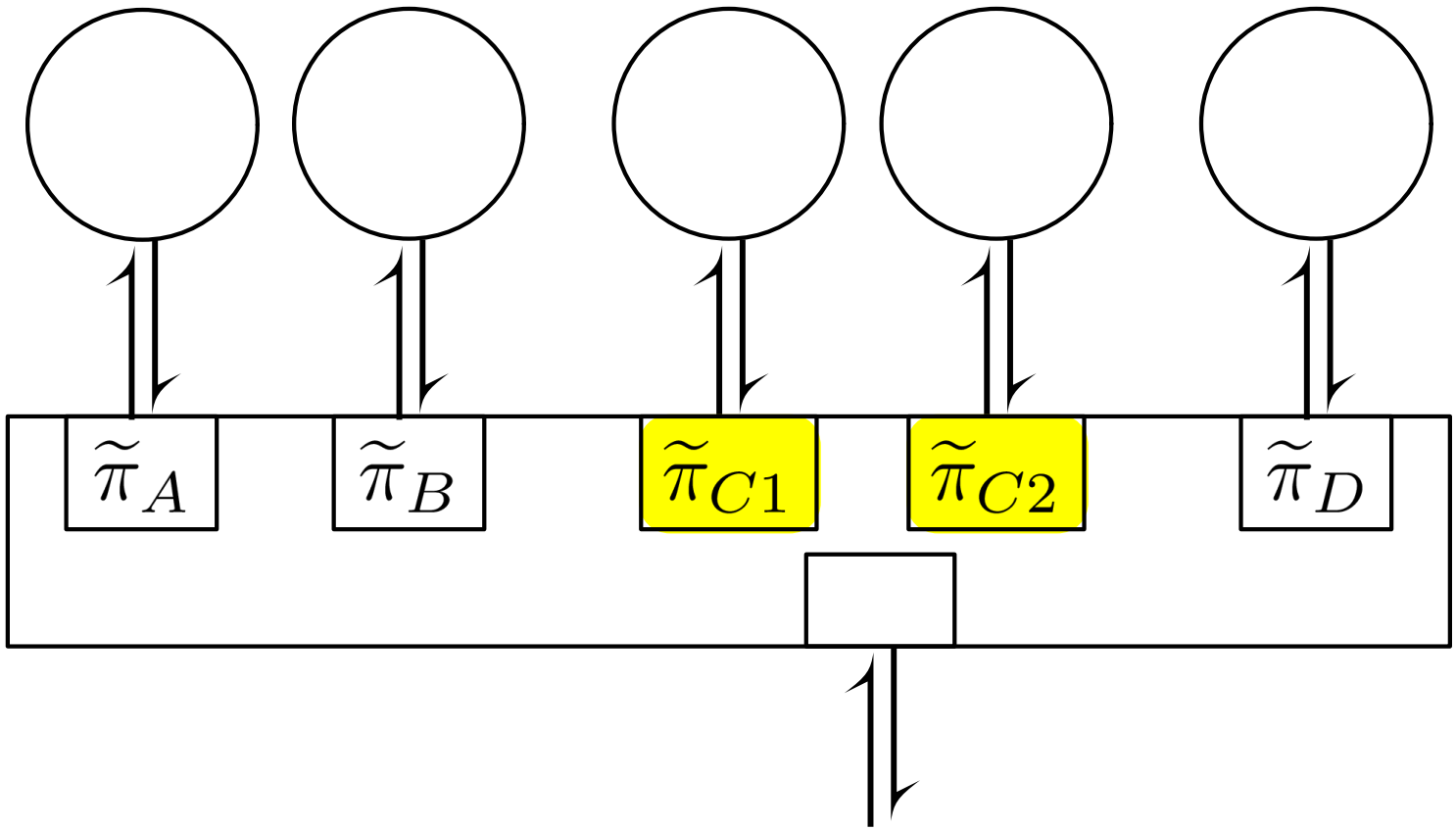
Layers make I/O Uniform



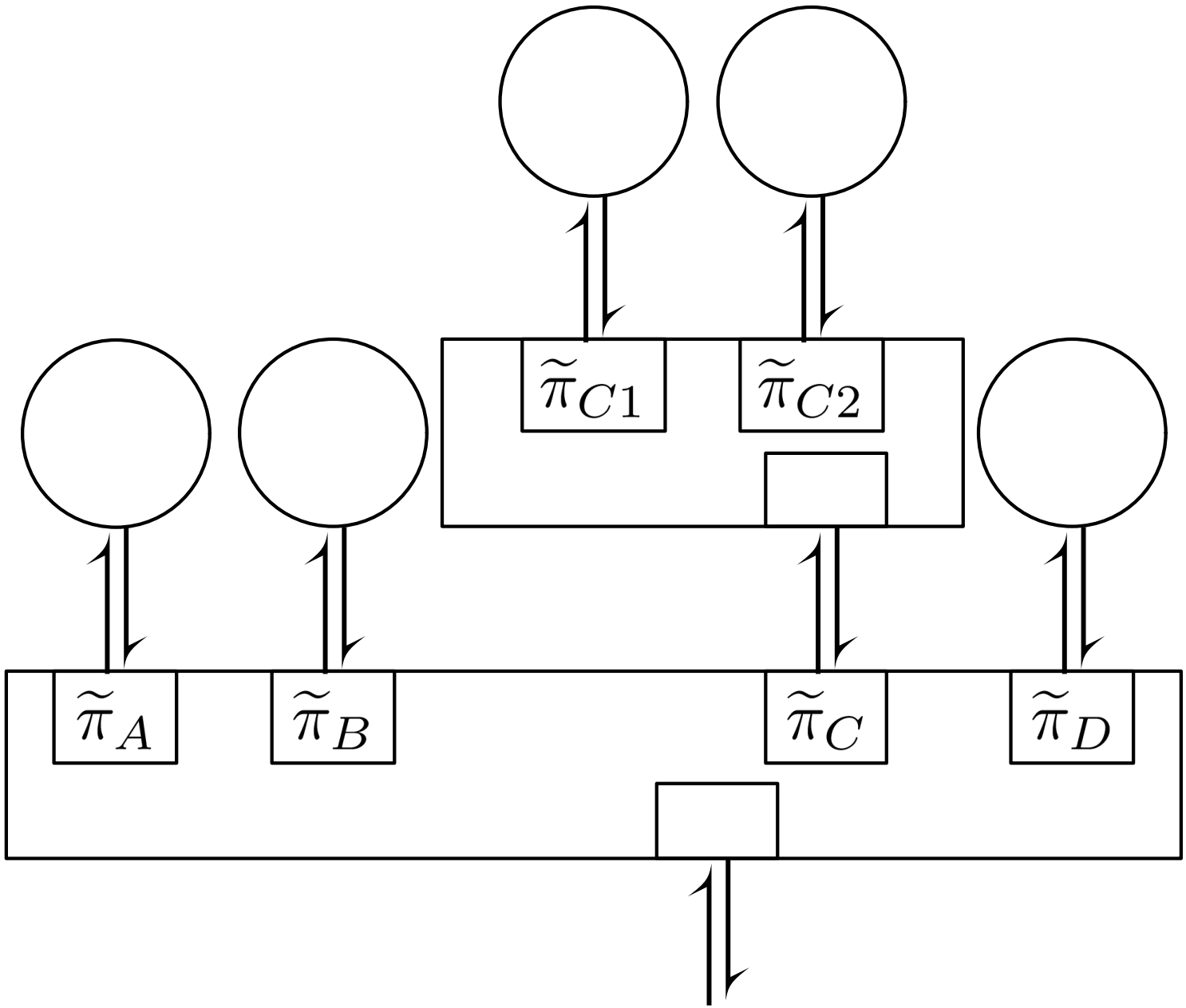
Layers Scope Conversations



Layers Scope Conversations

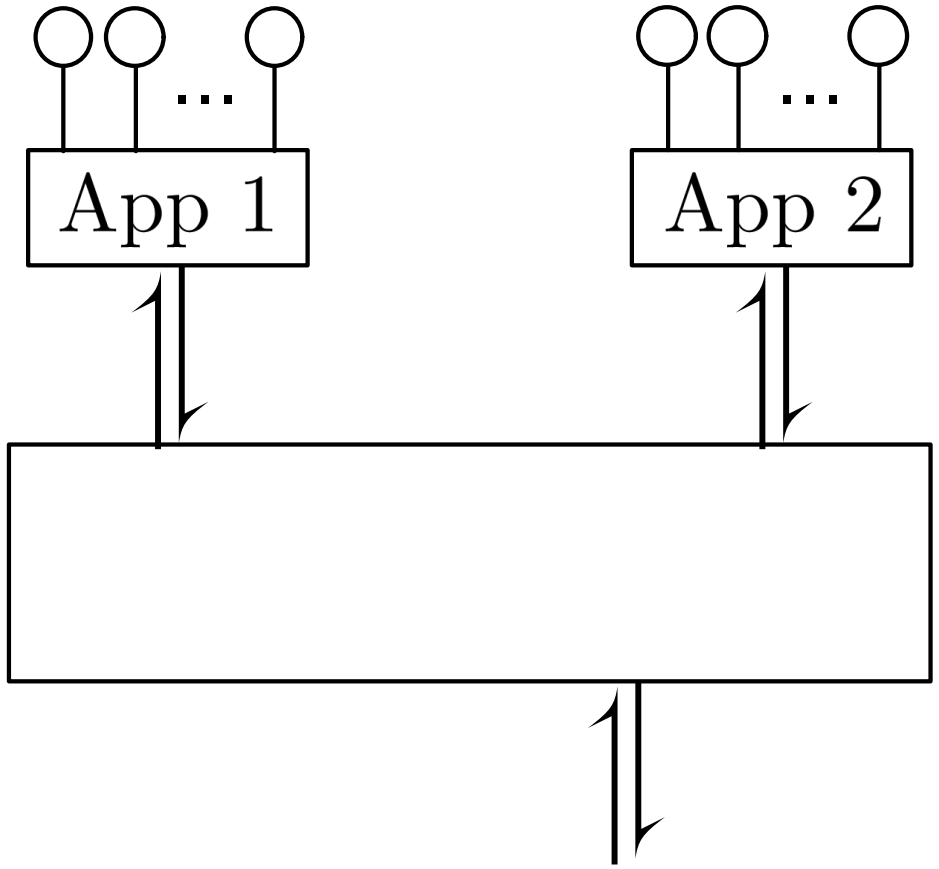


Layers Scope Conversations

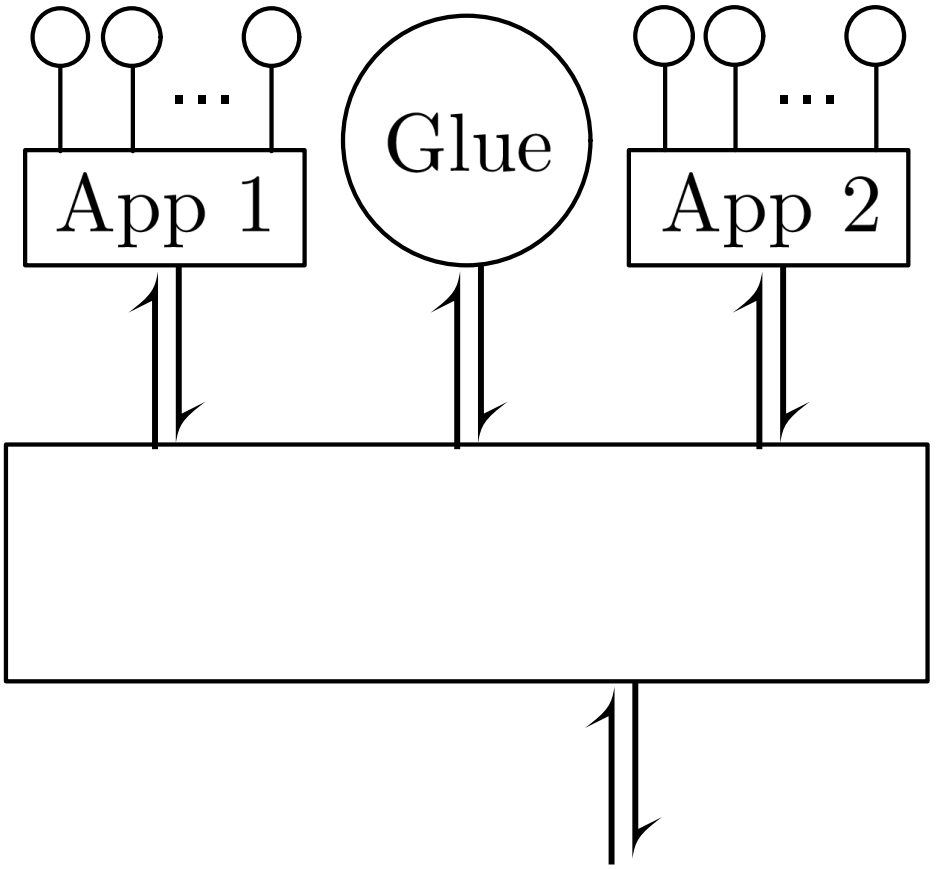


Layers Compose

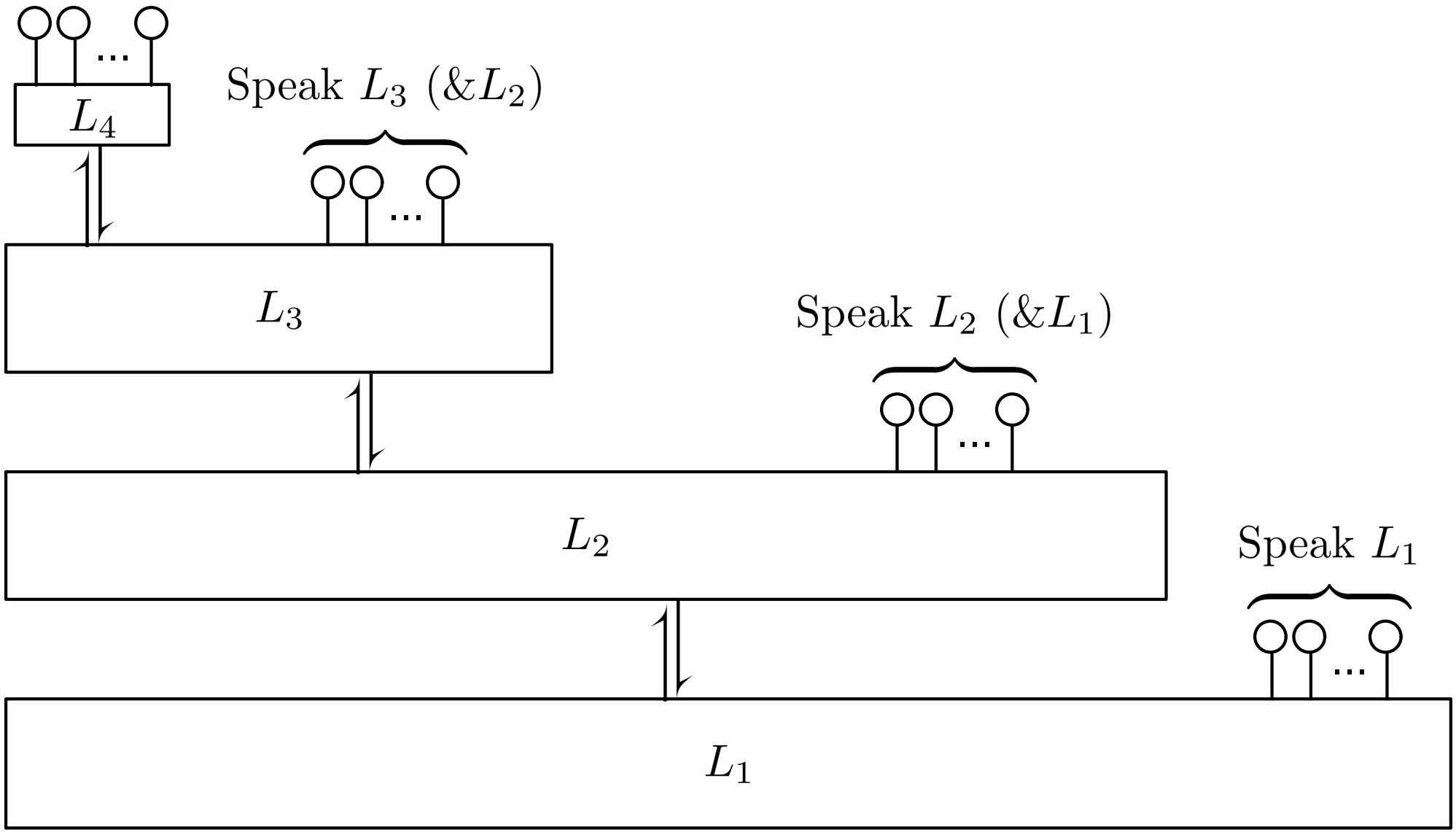
Layers Compose



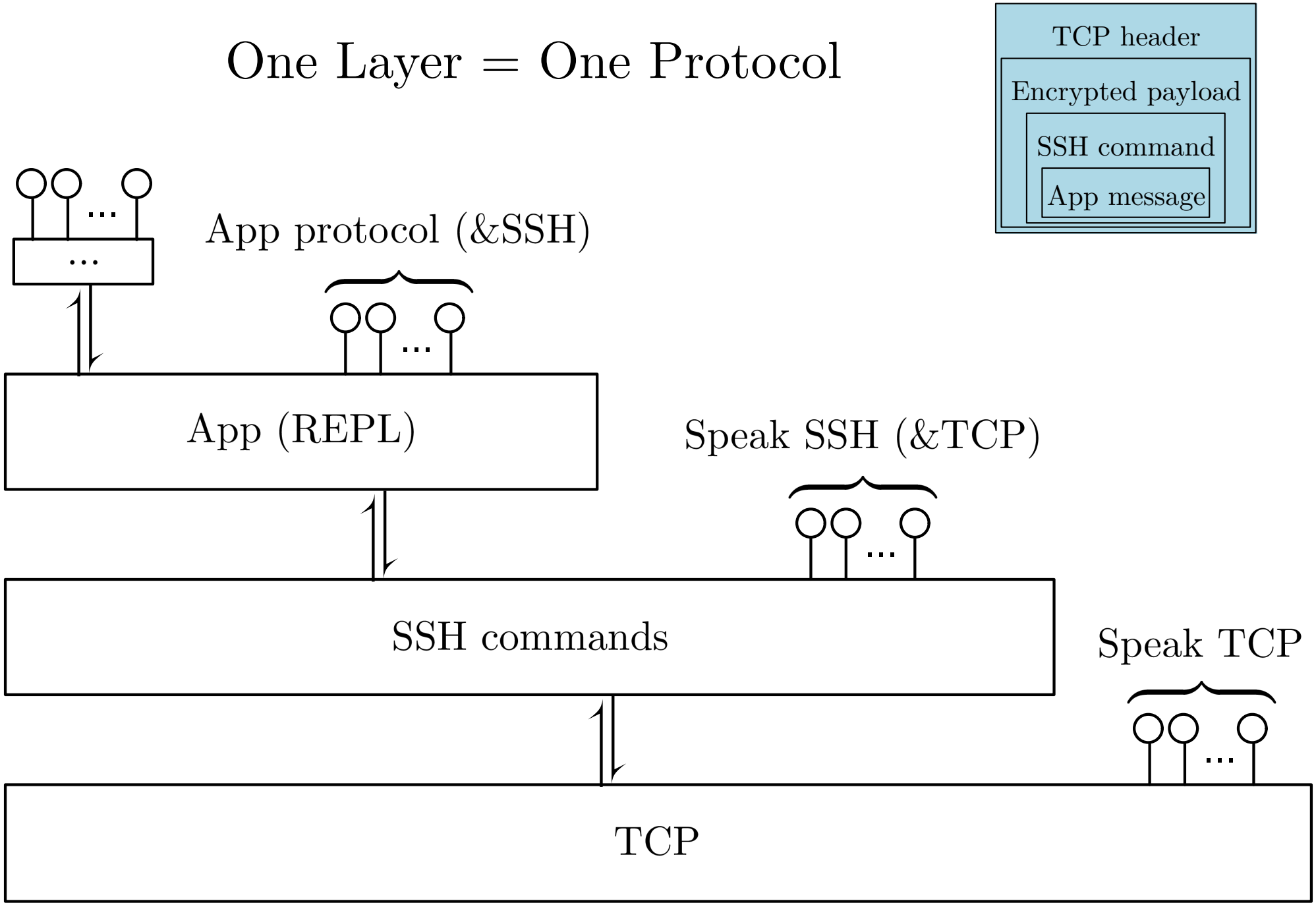
Layers Compose



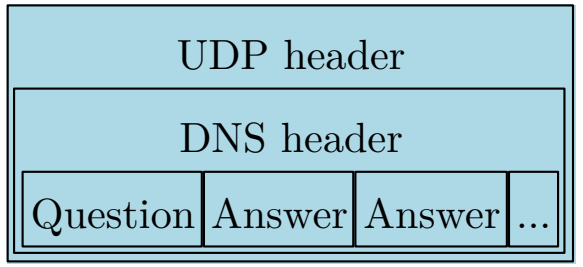
One Layer = One Protocol

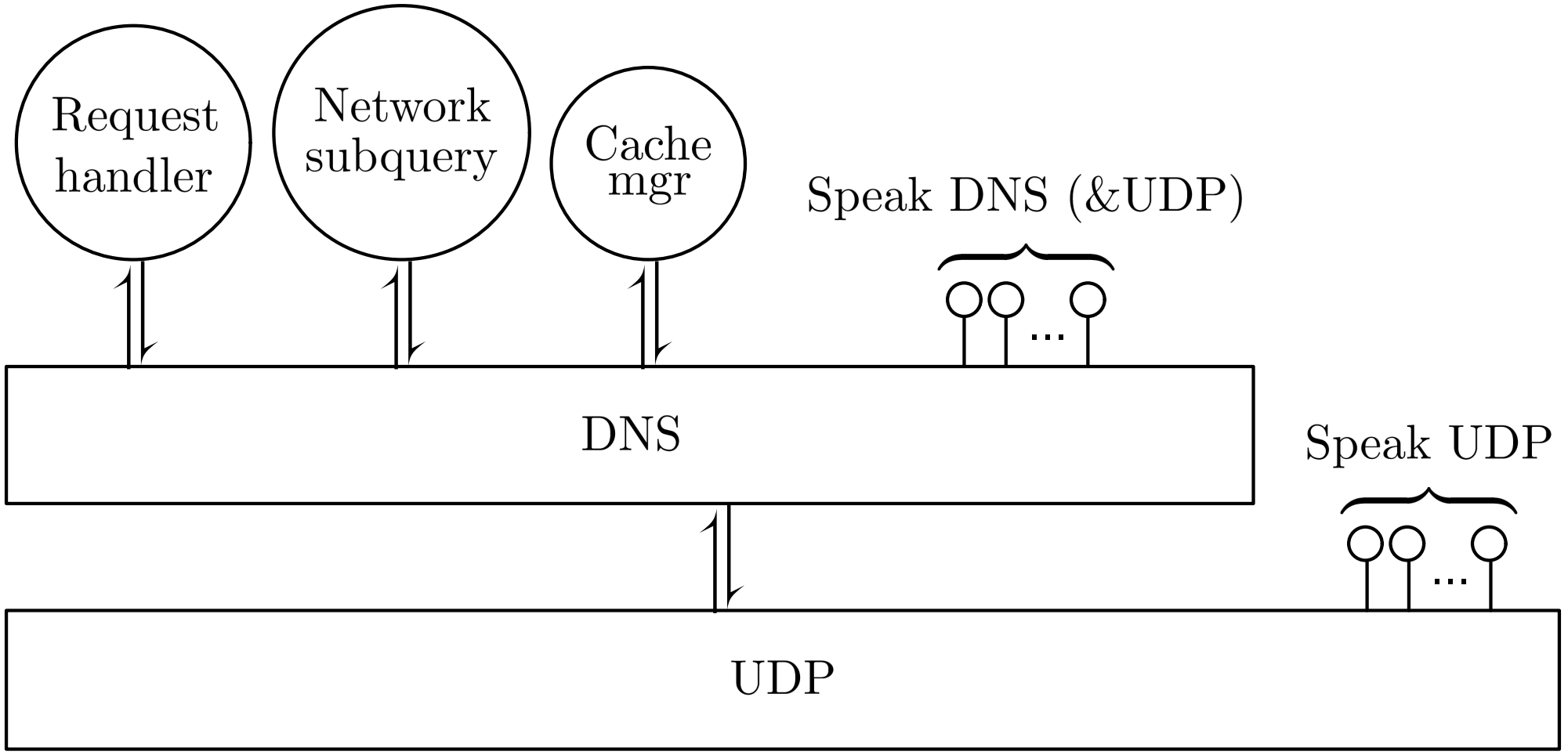


One Layer = One Protocol

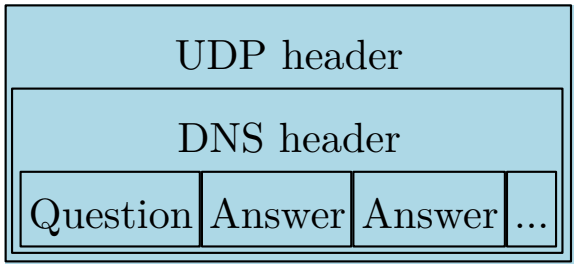


One Layer = One Protocol

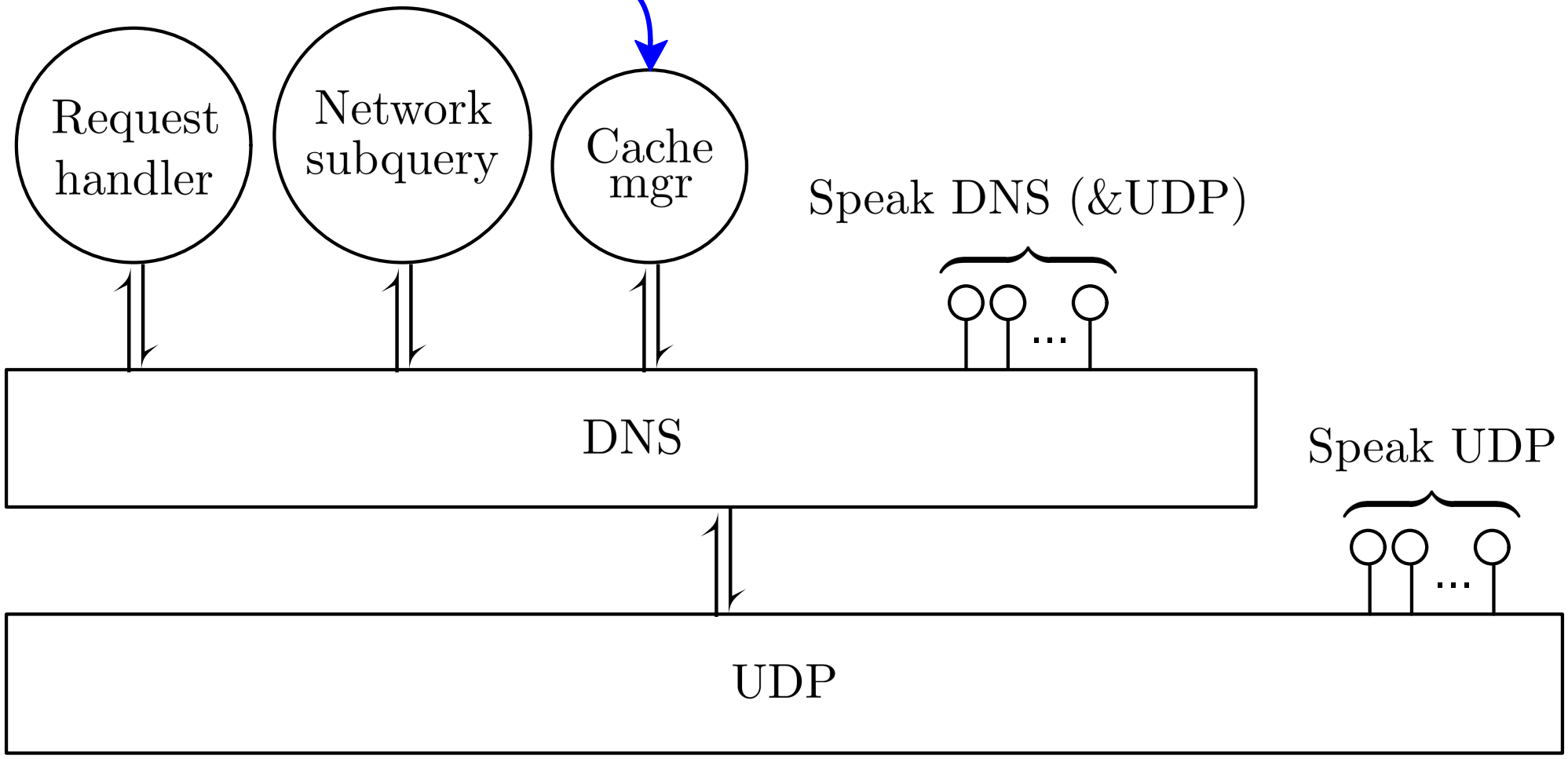




One Layer = One Protocol



Snoops via pub/sub to populate cache!



Interests — Local sub — Local adv

$$\pi \quad = \quad (p) \quad | \quad \langle p \rangle$$

Messages — Send *locally*

$$m \quad = \quad \langle v \rangle$$

Interests

Local sub

Local adv

Outside interest

$$\pi \quad = \quad (p) \quad | \quad \langle p \rangle \quad | \quad \lfloor \pi$$

Messages

Send *locally*

Send *outside*

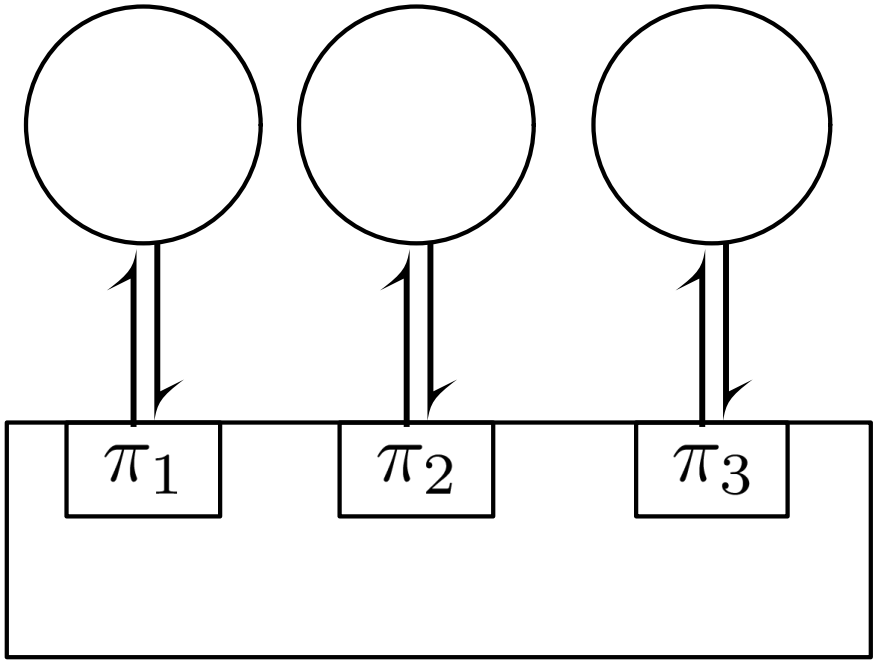
$$m \quad = \quad \langle v \rangle \quad | \quad \lfloor m$$

Interests

$$\pi \quad = \quad (p) \quad | \quad \langle p \rangle \quad | \quad \lfloor \pi$$

Messages

$$m \quad = \quad \langle v \rangle \quad | \quad \lfloor m$$

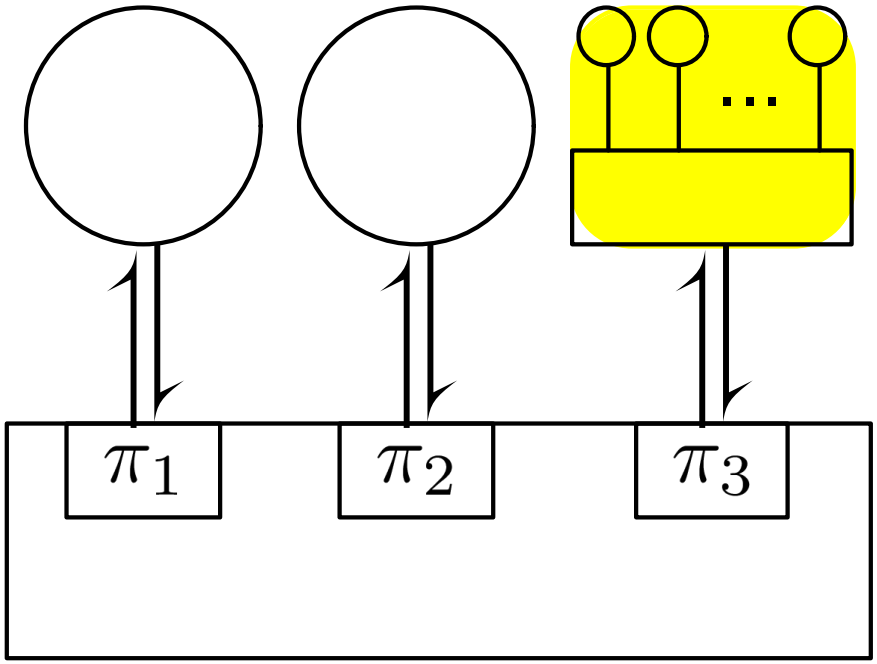


Interests — Local sub — Local adv — Outside interest

$$\pi \quad = \quad (p) \quad | \quad \langle p \rangle \quad | \quad \lfloor \pi$$

Messages — Send *locally* — Send *outside*

$$m \quad = \quad \langle v \rangle \quad | \quad \lfloor m$$

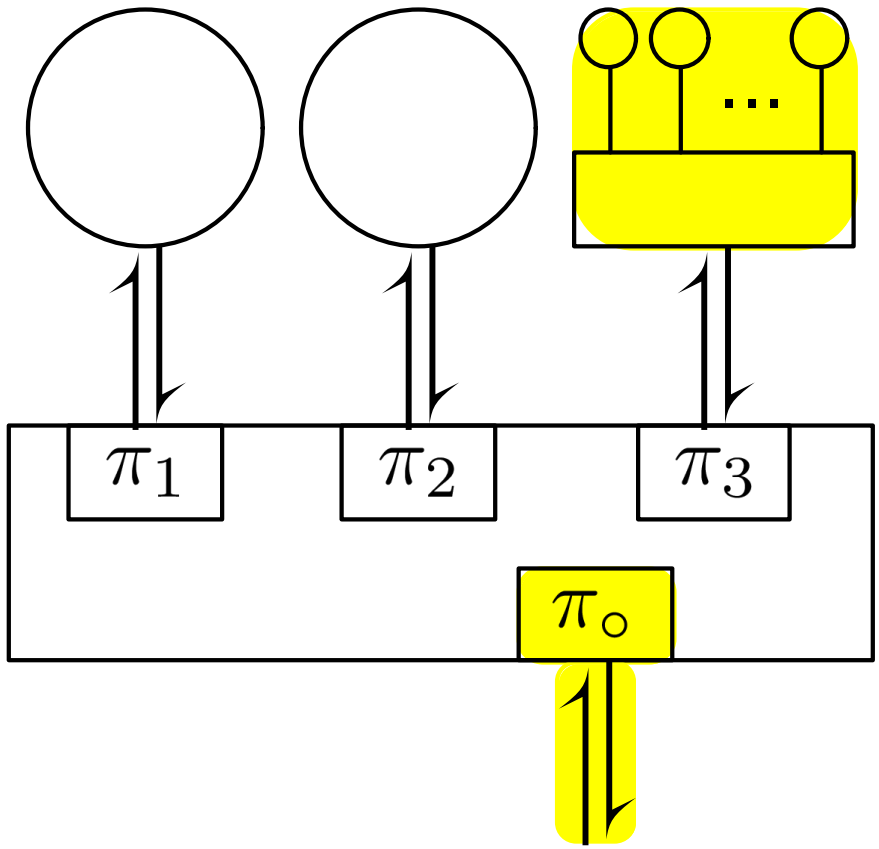


Interests — Local sub — Local adv — Outside interest

$$\pi \quad = \quad (p) \quad | \quad \langle p \rangle \quad | \quad \lfloor \pi$$

Messages — Send *locally* — Send *outside*

$$m \quad = \quad \langle v \rangle \quad | \quad \lfloor m$$



Logging: Requirements Scorecard

Route log entries from producers to consumers	☑ pub/sub
Consumers filter log messages	☑ pub/sub
Decouple producers from consumers	☑ pub/sub
Avoid shared-state explosion	☑ pub/sub
Discovery of logging service	☑ routing events
Only produce if someone's listening	☑ routing events
Alert when a producer crashes/exits	☑ routing events
Uniform treatment of I/O	☑ layering

Logging: Requirements Scorecard

Route log entries from producers to consumers	☑ pub/sub
Consumers filter log messages	☑ pub/sub
Decouple producers from consumers	☑ pub/sub
Avoid shared-state explosion	☑ pub/sub
Discovery of logging service	☑ routing events
Only produce if someone's listening	☑ routing events
Alert when a producer crashes/exits	☑ routing events
Uniform treatment of I/O	☑ layering
+ great additional benefits from layering	☑

PART V: Conclusions

Marketplace

Typed Racket

Minimart

Racket

JS-Marketplace

Javascript

Marketplace

Typed Racket

DNS server (UDP)

SSH server (TCP)

Chat server

Echo server

Minimart

Racket

Websocket driver

Generic msg broker

JS-Marketplace

Javascript

Websocket driver

DOM driver

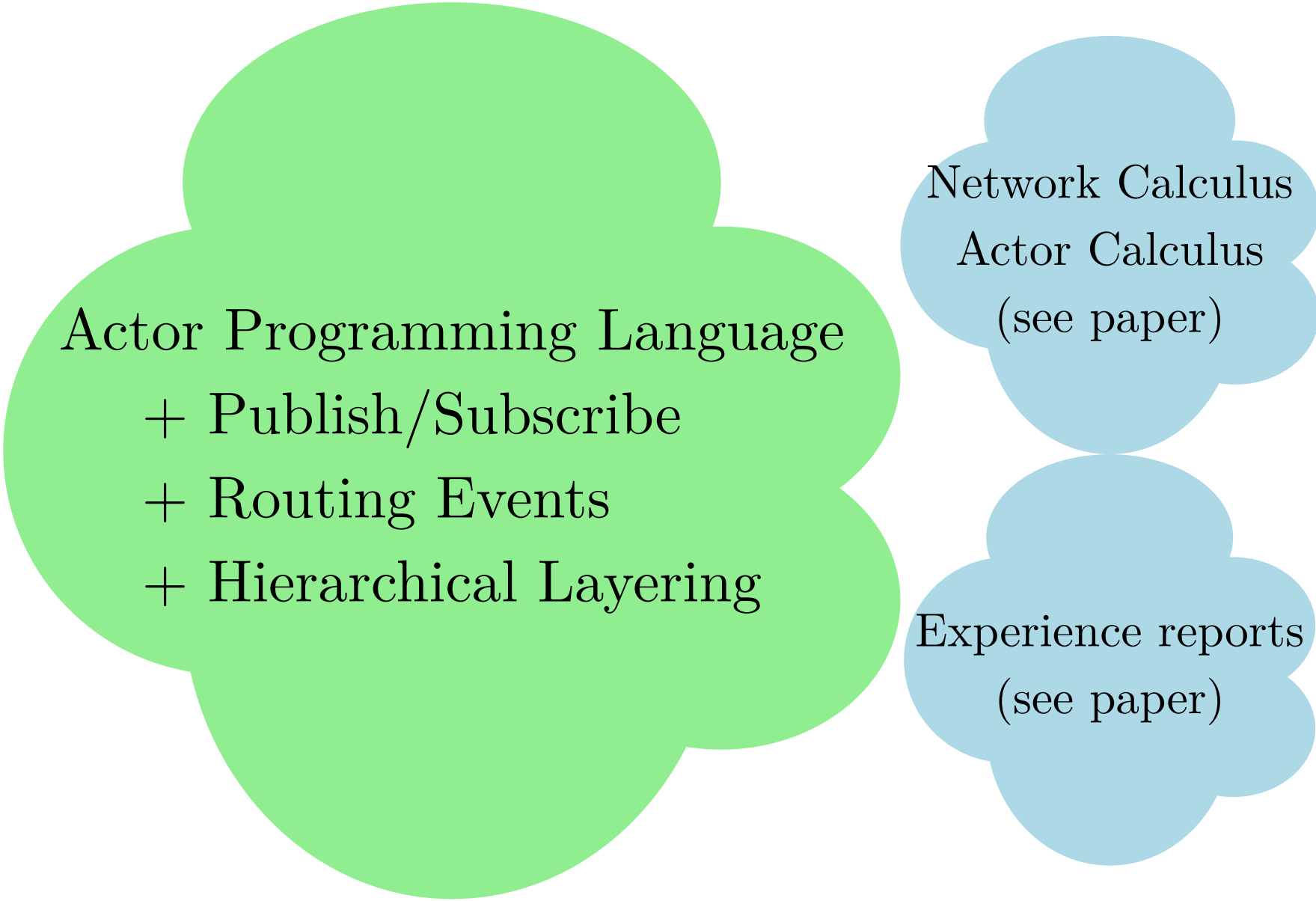
jQuery driver

Chat + roster

GUI composition

Marketplace	Minimart	JS-Marketplace
Typed Racket	Racket	Javascript
DNS server (UDP)	Websocket driver	Websocket driver
SSH server (TCP)	Generic msg broker	DOM driver
Chat server		jQuery driver
Echo server		Chat + roster
		GUI composition

Details and experience report in the paper!


Actor Programming Language
+ Publish/Subscribe
+ Routing Events
+ Hierarchical Layering
Network Calculus
Actor Calculus
(see paper)
Experience reports
(see paper)

Thank you!

Actor Programming Language
+ Publish/Subscribe
+ Routing Events
+ Hierarchical Layering

Network Calculus
Actor Calculus
(see paper)

Experience reports
(see paper)

http://www.ccs.neu.edu/home/tonyg/marketplace/