Imperial College London

Enforcing User Privacy in Web Applications using Erlang

Web 2.0 Security & Privacy (W2SP) 2010
May 20, Berkeley, California, USA

Ioannis Papagiannis, Matteo Migliavacca, Peter Pietzuch
Department of Computing
Imperial College London

David Eyers, Jean Bacon
Computer Laboratory
University of Cambridge

Brian Shand
CBCU
National Health Service

Imperial College London

User Privacy in Web Applications

- Which is longer, the United States Constitution or Facebook's Privacy Policy?
  - Facebook's Privacy Policy: 5,830 words
  - United States Constitution: 4,543 words

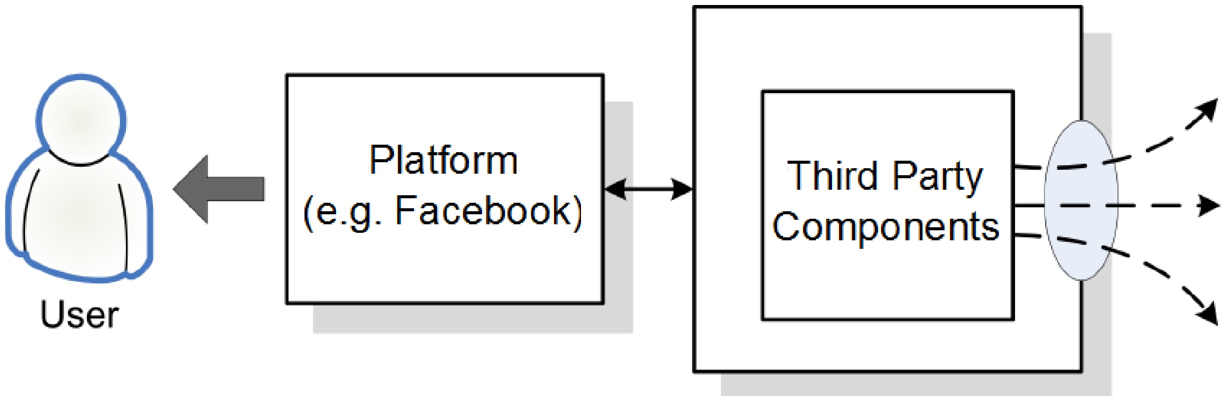
[NYT, May 12, 2010]

- Twitter 0 followers bug
  - Tweet "accept," followed by "@" and user name
  - The other user starts following you automatically (!)

[Official Twitter Blog, May 10, 2010]

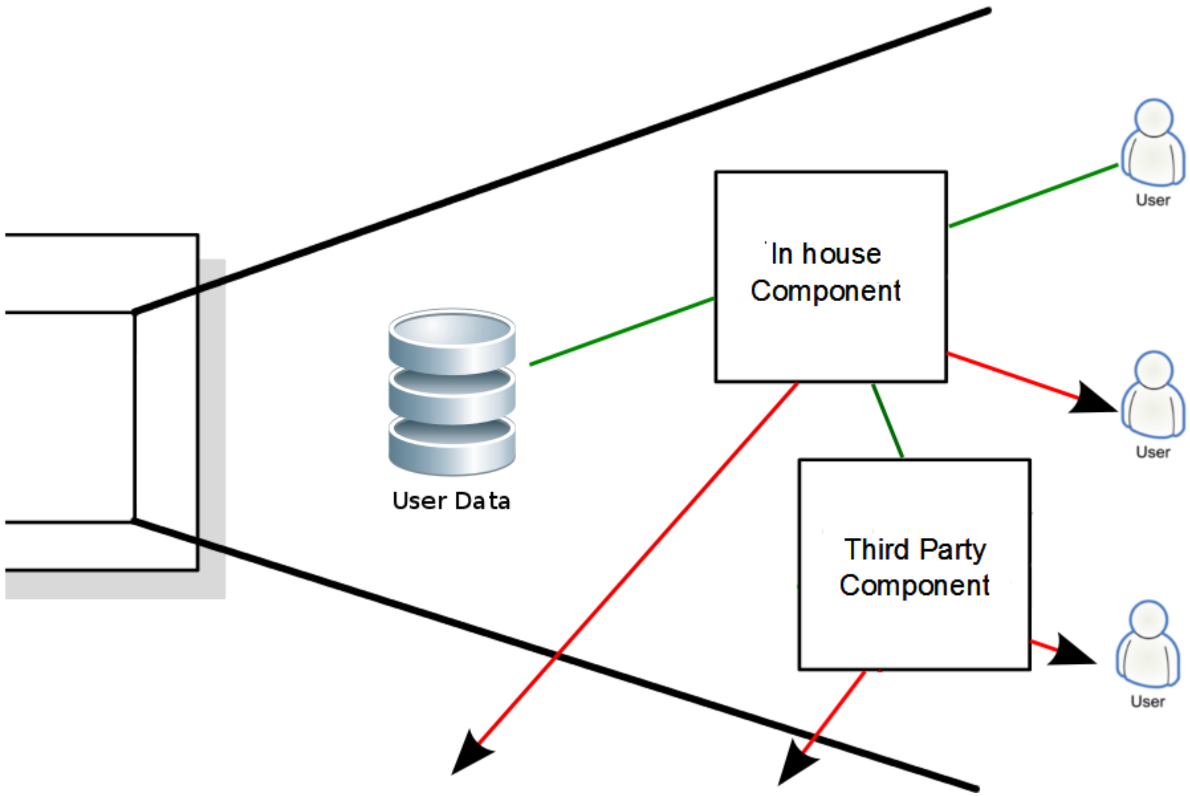
Imperial College London

User Privacy in Web Applications



- User data privacy must be guaranteed independently of the application's functional correctness

User Privacy in Web Applications



- ➢Code should access only relevant user data and keep them isolated from other users' data

Use Case: Privacy in Microblogging

A microblogging system should guarantee:

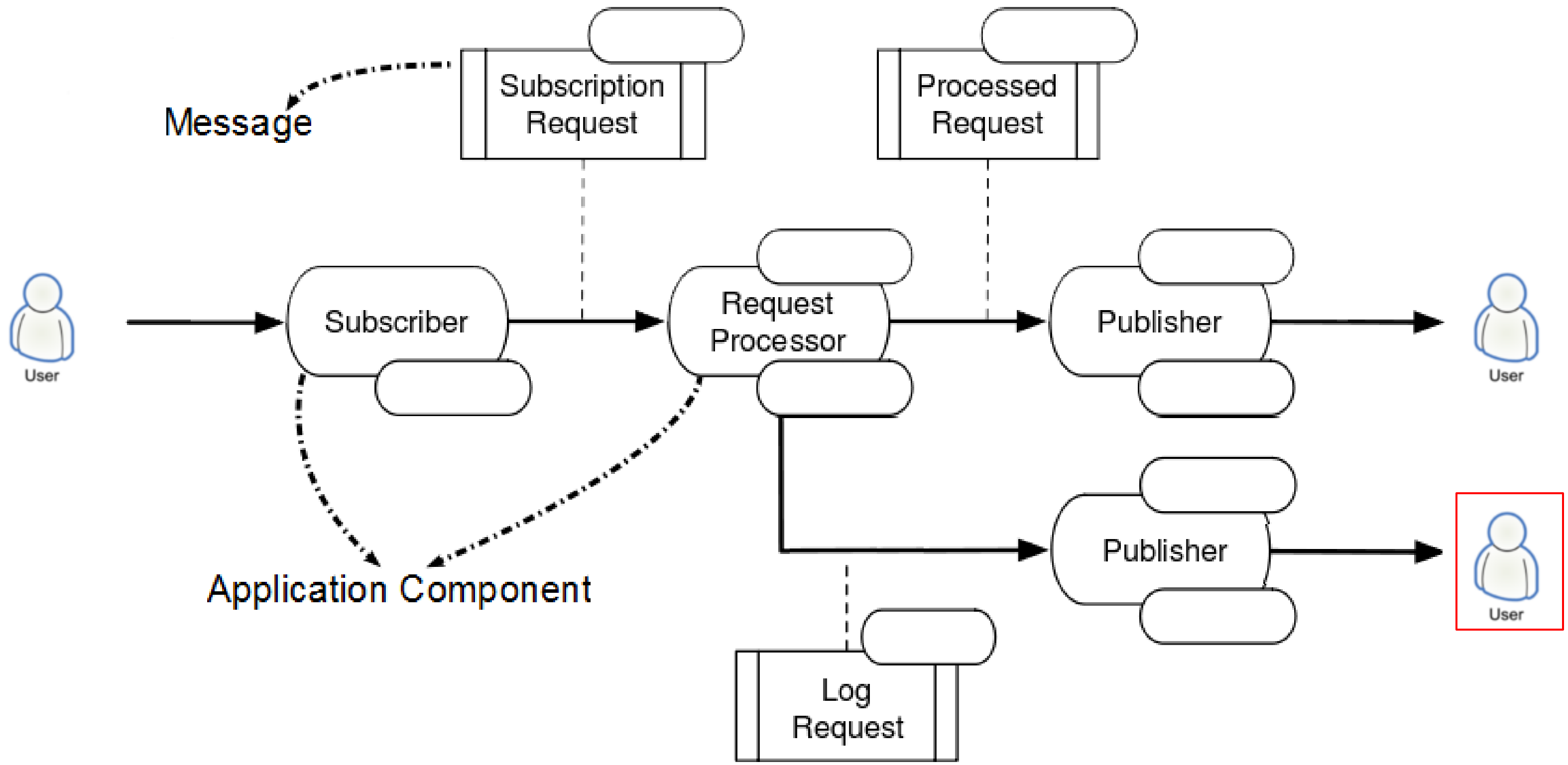
1. *Messages from a publisher component shall be delivered only to authorised subscribers' components.*
   [User A's messages will only go to Users B and C]

2. *Authorised subscribers shall not be disclosed to any other publisher or subscriber component.*
   [User B will not know about User C]

3. *Subscription authorisation requests from a subscribing component shall be delivered only to the relevant publisher's component.*
   [Only User A can authorise a new User D]

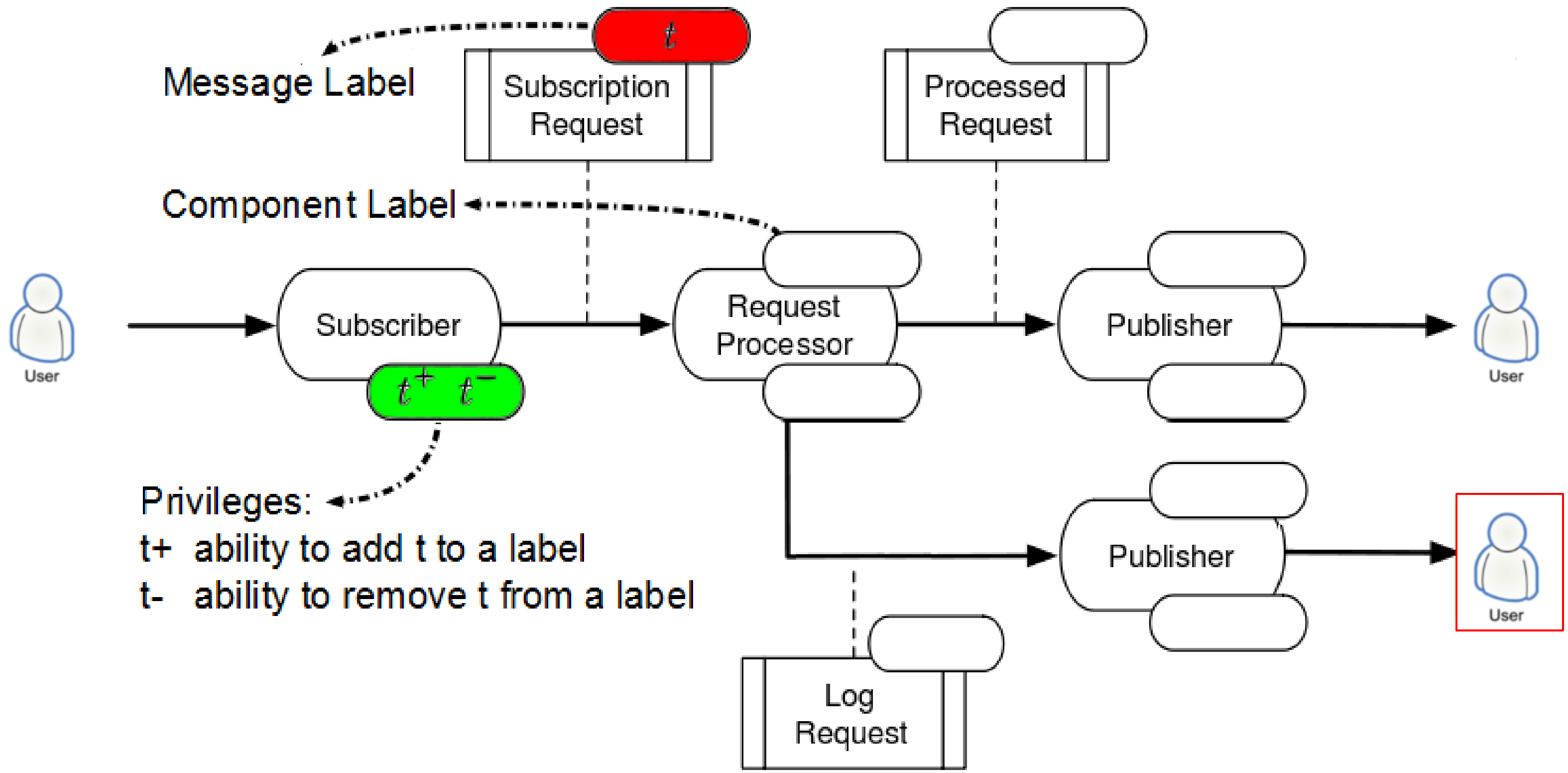
Imperial College London

IFC for Microblogging



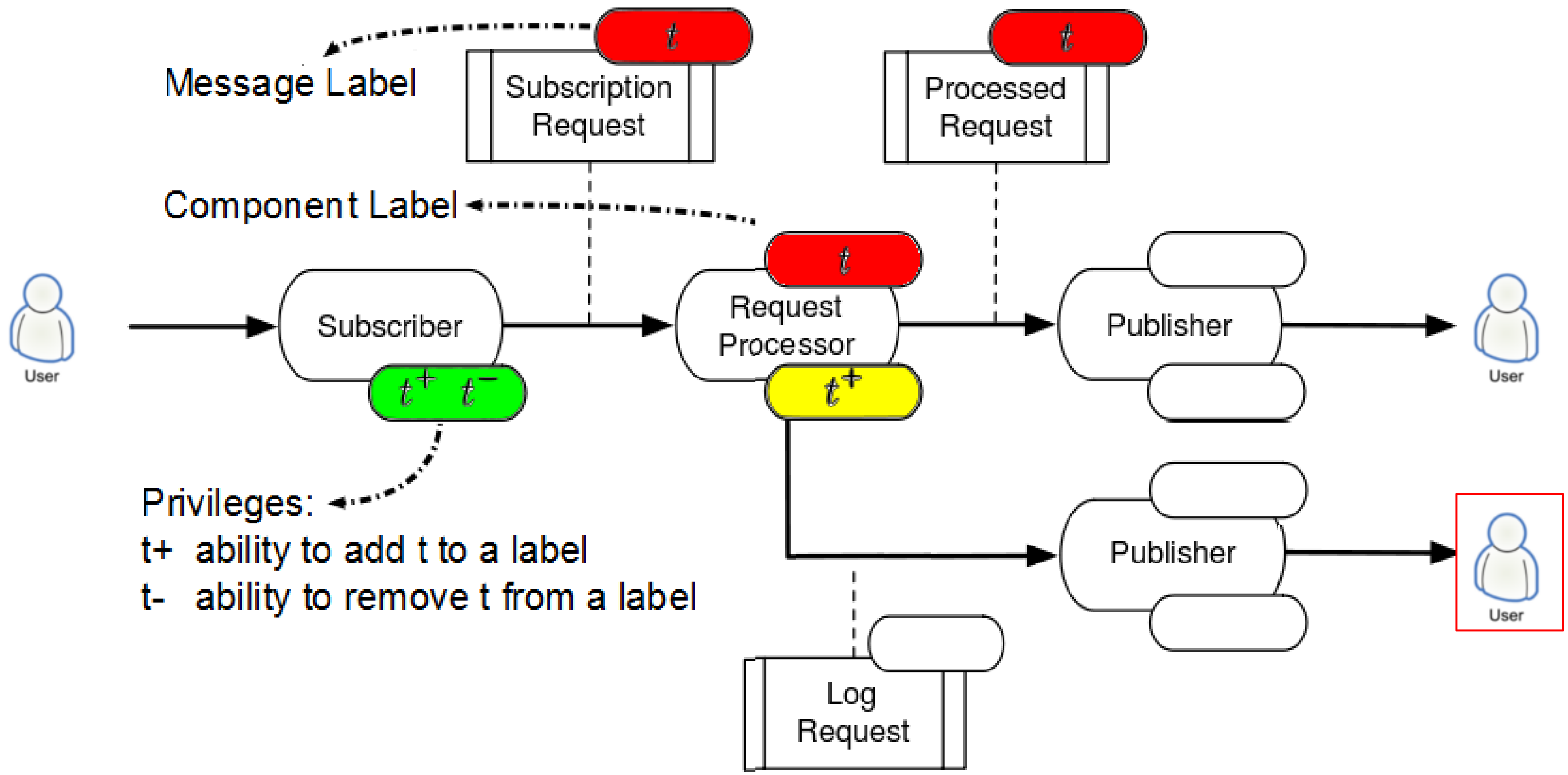
Imperial College London

IFC for Microblogging



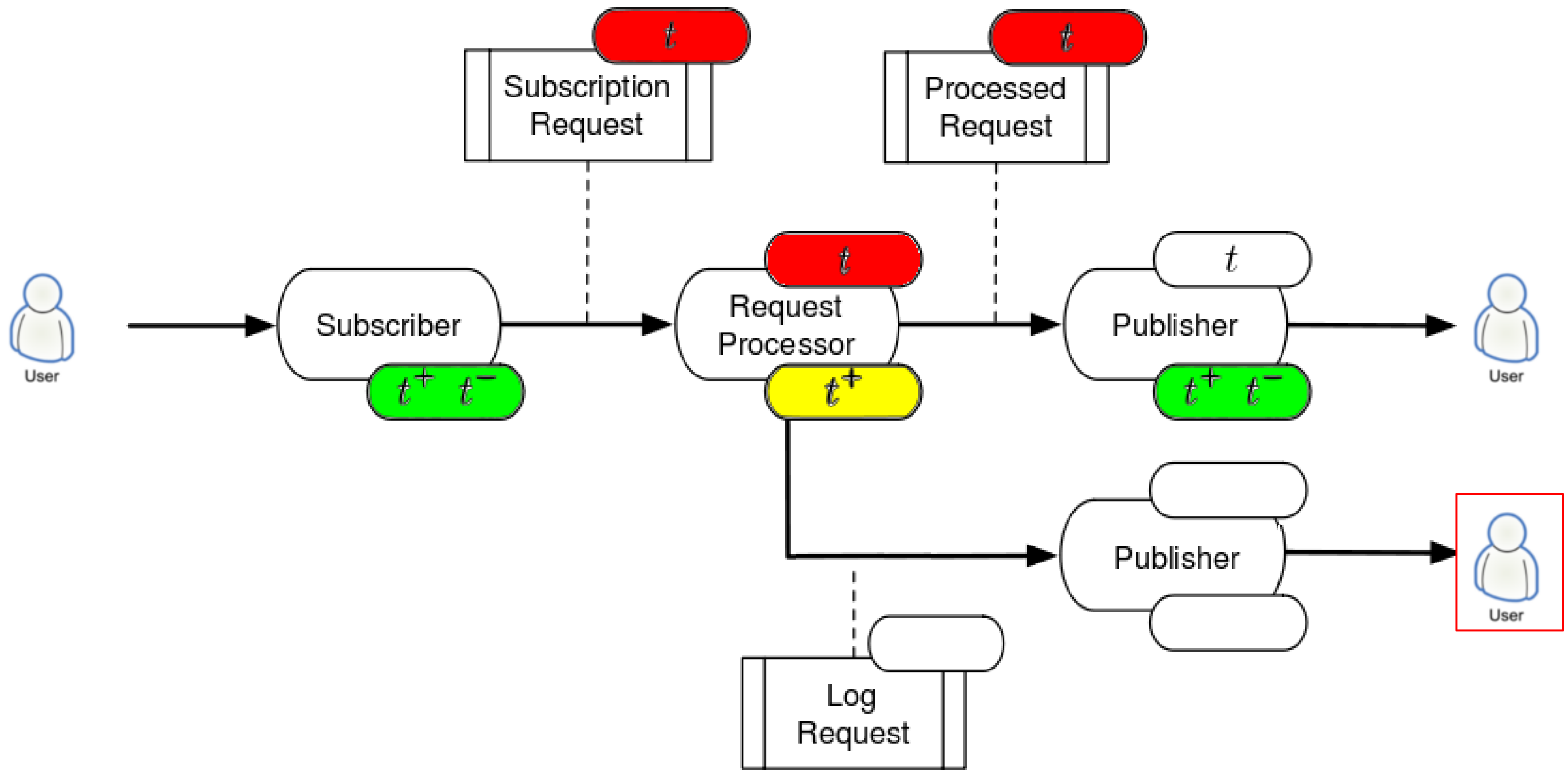
Privileges:
t+ ability to add t to a label
t- ability to remove t from a label

IFC for Microblogging

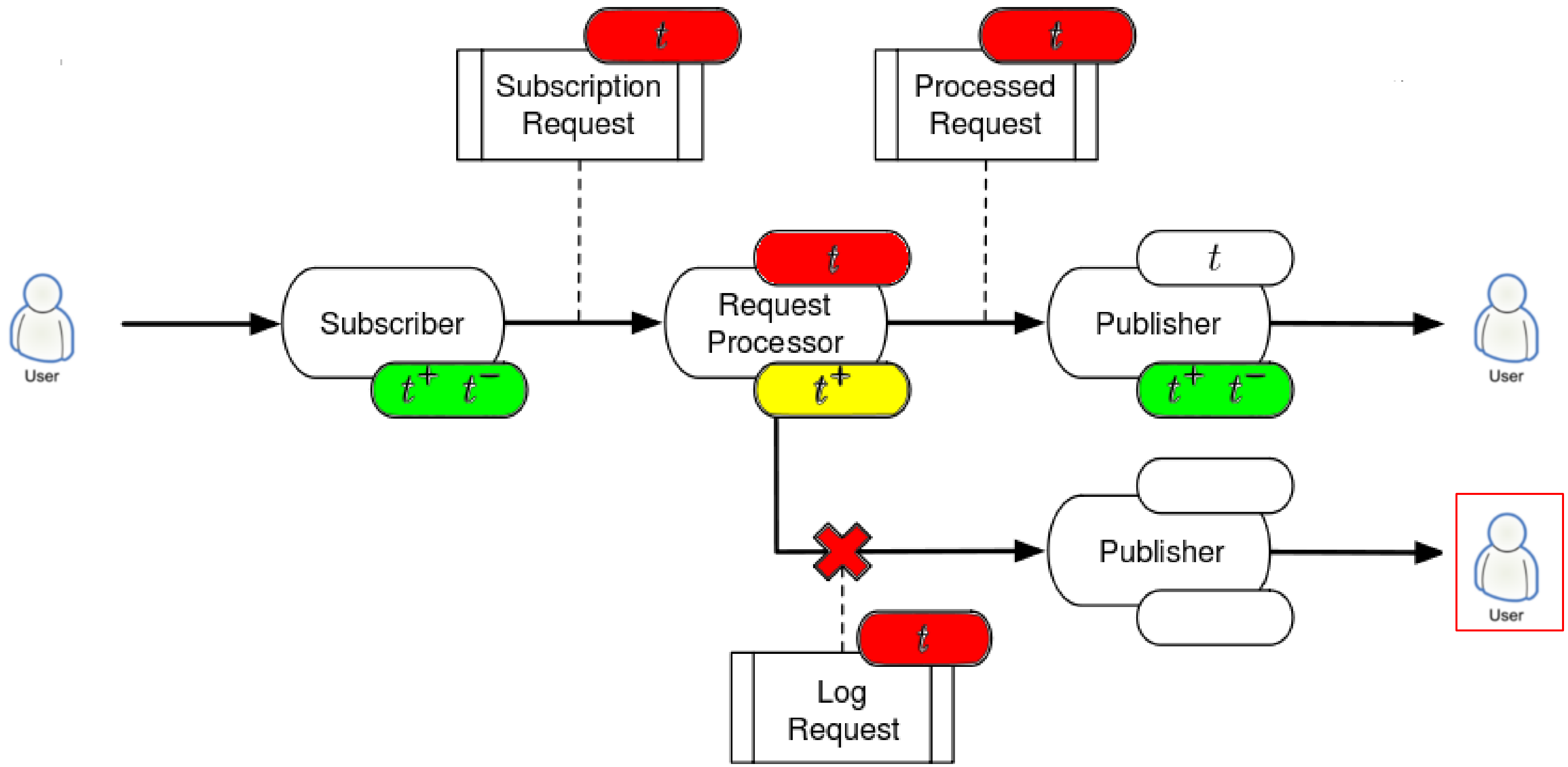


Imperial College London

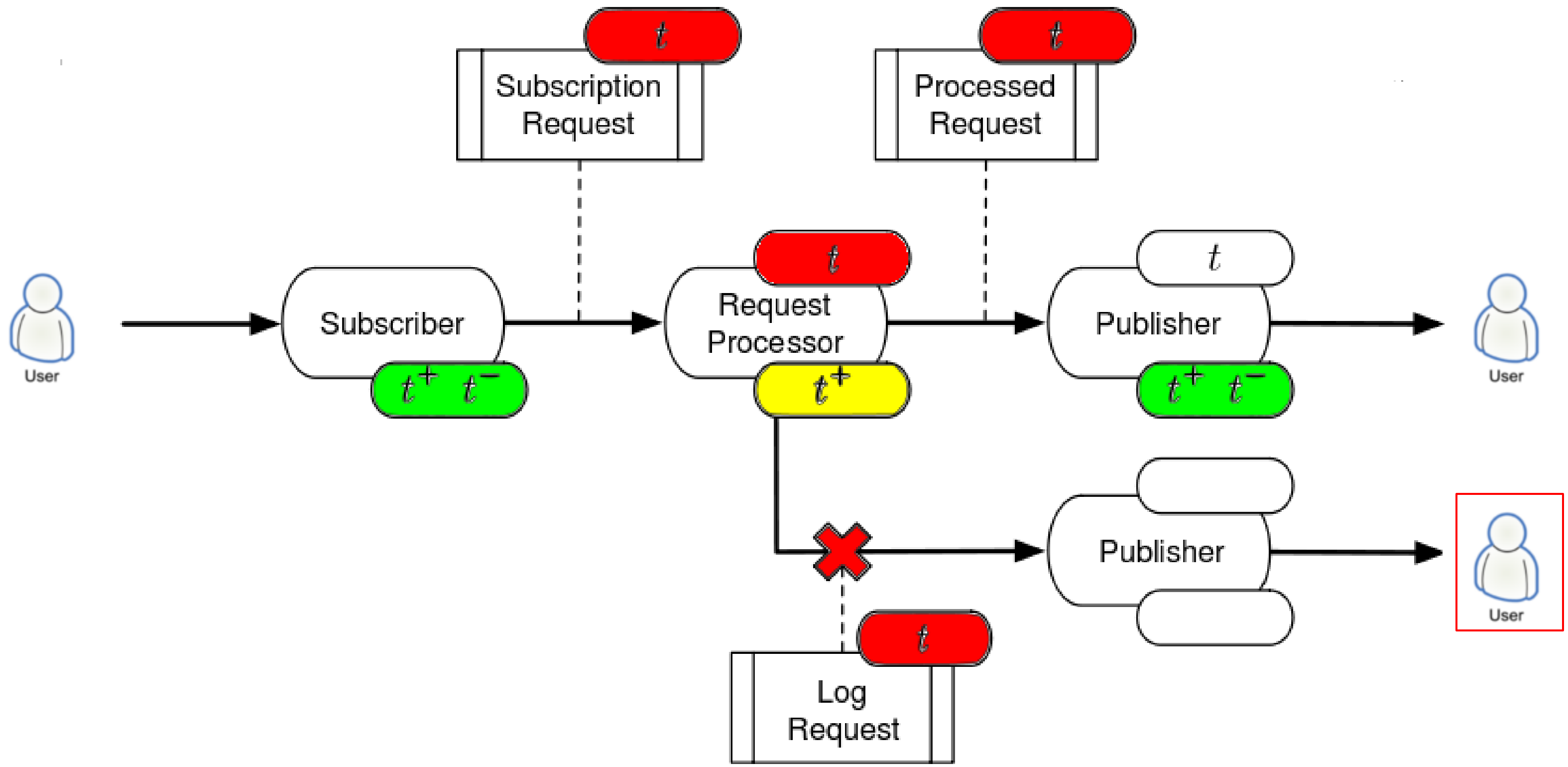
IFC for Microblogging



IFC for Microblogging



IFC for Microblogging

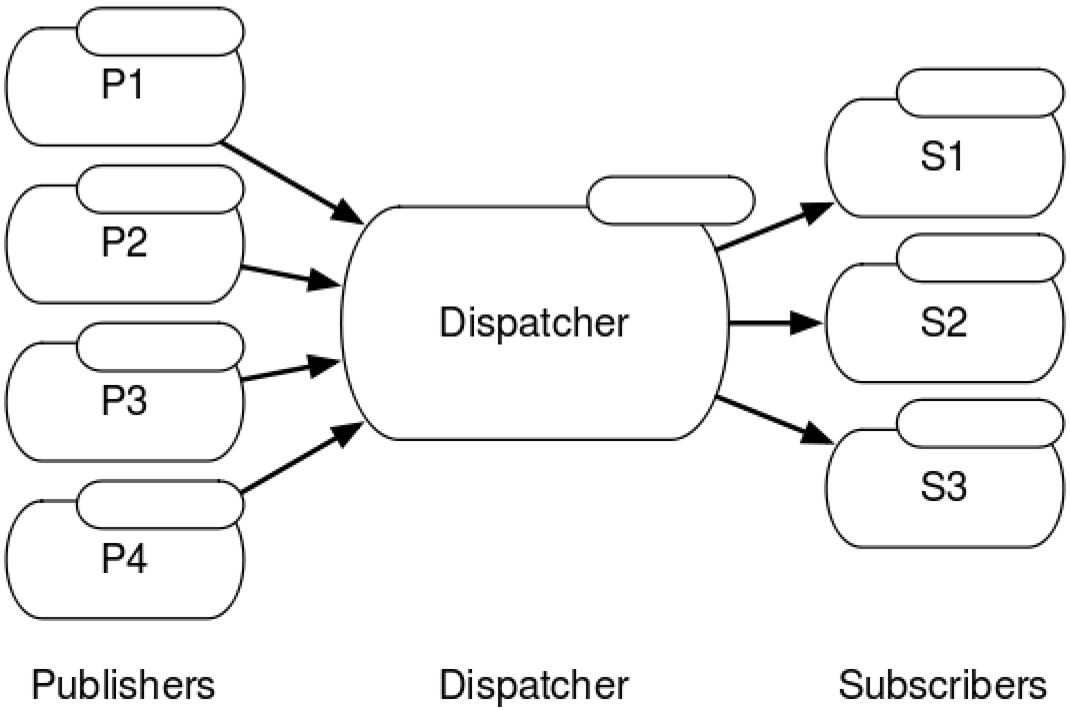


What happens when data belonging to different users has to be processed by a <u>single</u> component?

Imperial College London

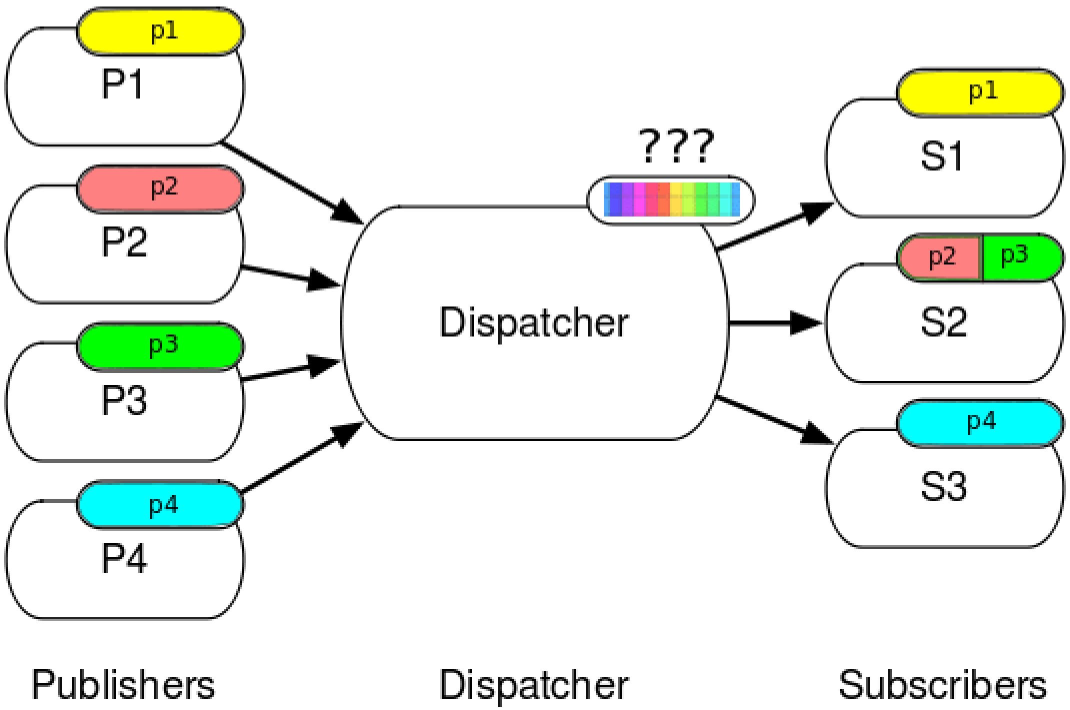
Microblogging: The Dispatcher

Multiple publishing components have to use a single dispatcher to reach the relevant subscriber components



Microblogging: The Dispatcher

Multiple publishing components have to use a single dispatcher to reach the relevant subscriber components.



Solution

- Each User's data must be kept separate, but applications are usually monolithic
- Compartmentalize the application in multiple isolated components, one per user
- Granularity?

Solution

- Each User's data must be kept separate, but applications are usually monolithic
- Compartmentalize the application in multiple isolated components, one per user
- Granularity?

Language	Isolation	Issue
C	OS Processes	~100kB per process

Imperial College London

Solution

- Each User's data must be kept separate, but applications are usually monolithic
- Compartmentalize the application in multiple isolated components, one per user
- Granularity?

Language	Isolation	Issue
C	OS Processes	~100kB per process
Java	OS Threads	Limited isolation: static fields, object locks, runtime channels

Imperial College London

Solution

- Each User's data must be kept separate, but applications are usually monolithic
- Compartmentalize the application in multiple isolated components, one per user
- Granularity?

Language	Isolation	Issue
C	OS Processes	~100kB per process
Java	OS Threads	Limited isolation: static fields, object locks, runtime channels
PHP JavaScript	OS Processes	Spawning a new runtime on top of spawning a new OS process

Erlang

- Sequential Part:
  functional language, single assignment, dynamic typing
- Concurrency:
  share nothing concurrency, message passing

- Erlang is great for IFC
  - Isolation is free
  - Asynchronous message passing can be naturally combined with label checks
  - Processes are lightweight (~100B, runtime implementation)

Erlang: Example

- Sender Process:

```
test(0) -> done;
test(N) ->
   pid=spawn(primeTester),
   pid ! {calculate, self(), N},
   receive
      {result, Result}->
         io:format("~w",[Result])
   end,
   test(N-1)
 end.
```

Spawning processes is __fast__!

You ~~can~~ want to have __lots__ of them!

- Receiver Process:

```
primeTester() ->
  receive
    {calculate, Pid, Number} ->
       Result = isPrime(Number),
       Pid ! {result, Result}
  end.
```

Async message passing is the __only way__ of communication!*

Imperial College London

Supporting IFC in Erlang

- Attach labels to pids
- `new_tag()`

  creates a new tag for the calling process
- `spawn(TagsAdd, TagsRemove, ...)`

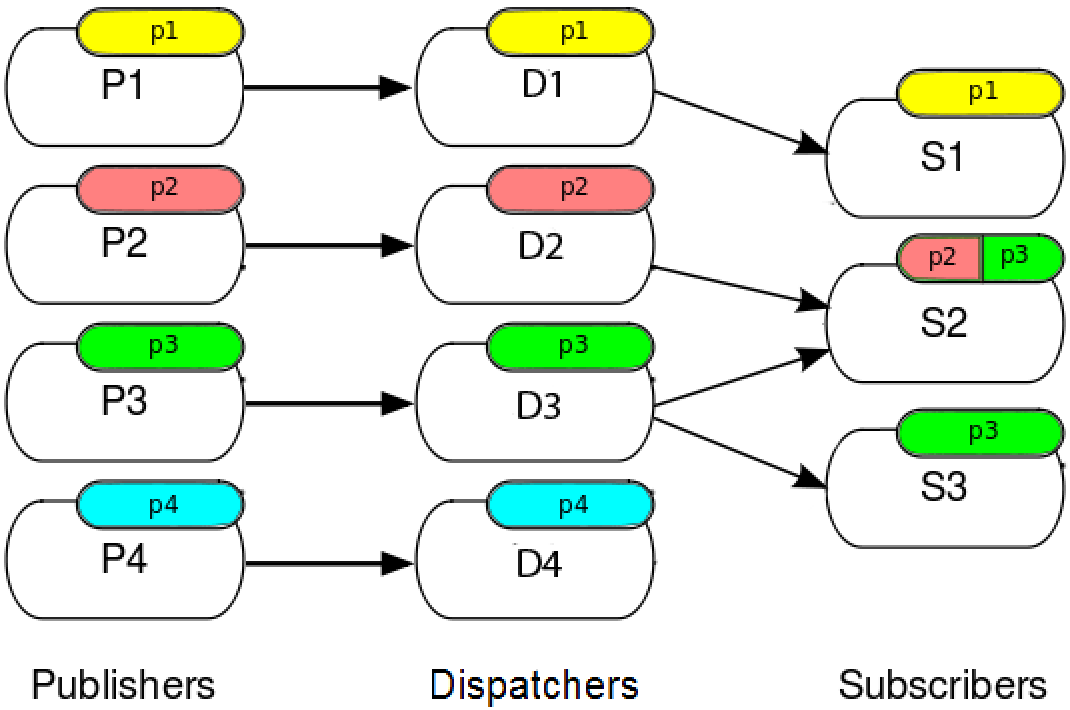
  changes the tags of the spawned process (≠ caller's tags)
- `send(TagsAdd, TagsRemove, ...)`

  changes the tags of the message (≠caller's tags)
  checks labels
- `delegate(PidReceiver, Tag, Type)`

  gives privileges over a tag to another process

Erlang for Microblogging I

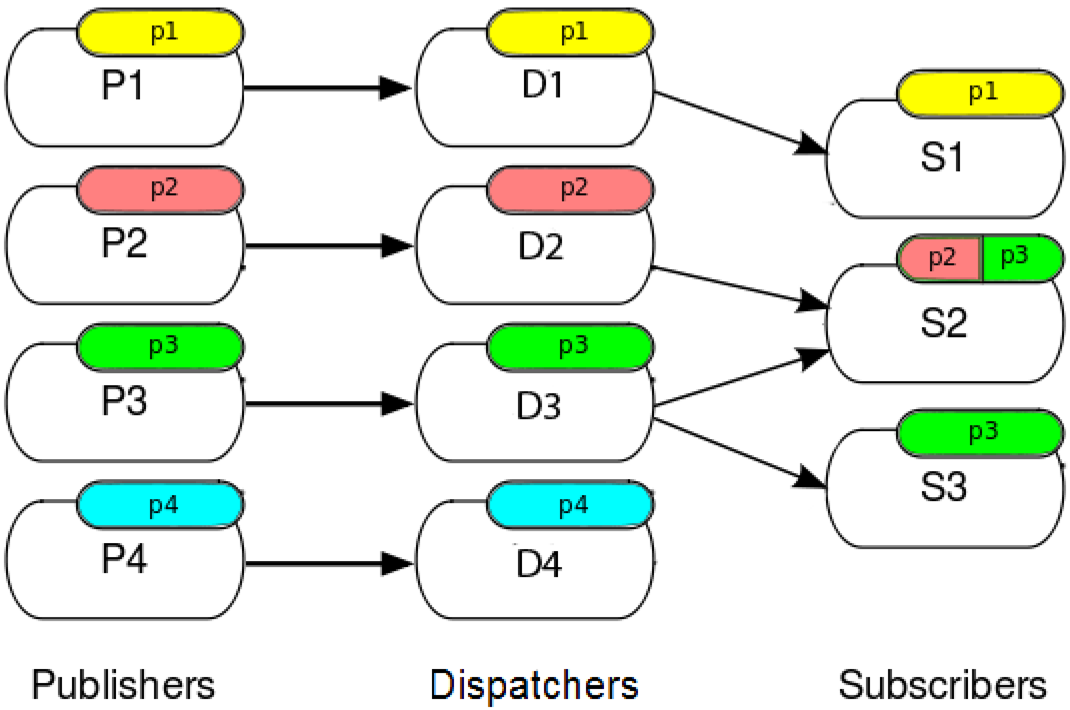
1. *Messages from a publisher shall be received only by authorised subscribers.*



(untrusted code)

Erlang for Microblogging I

2. *Authorised subscribers shall not be disclosed to any other publisher or subscriber.*

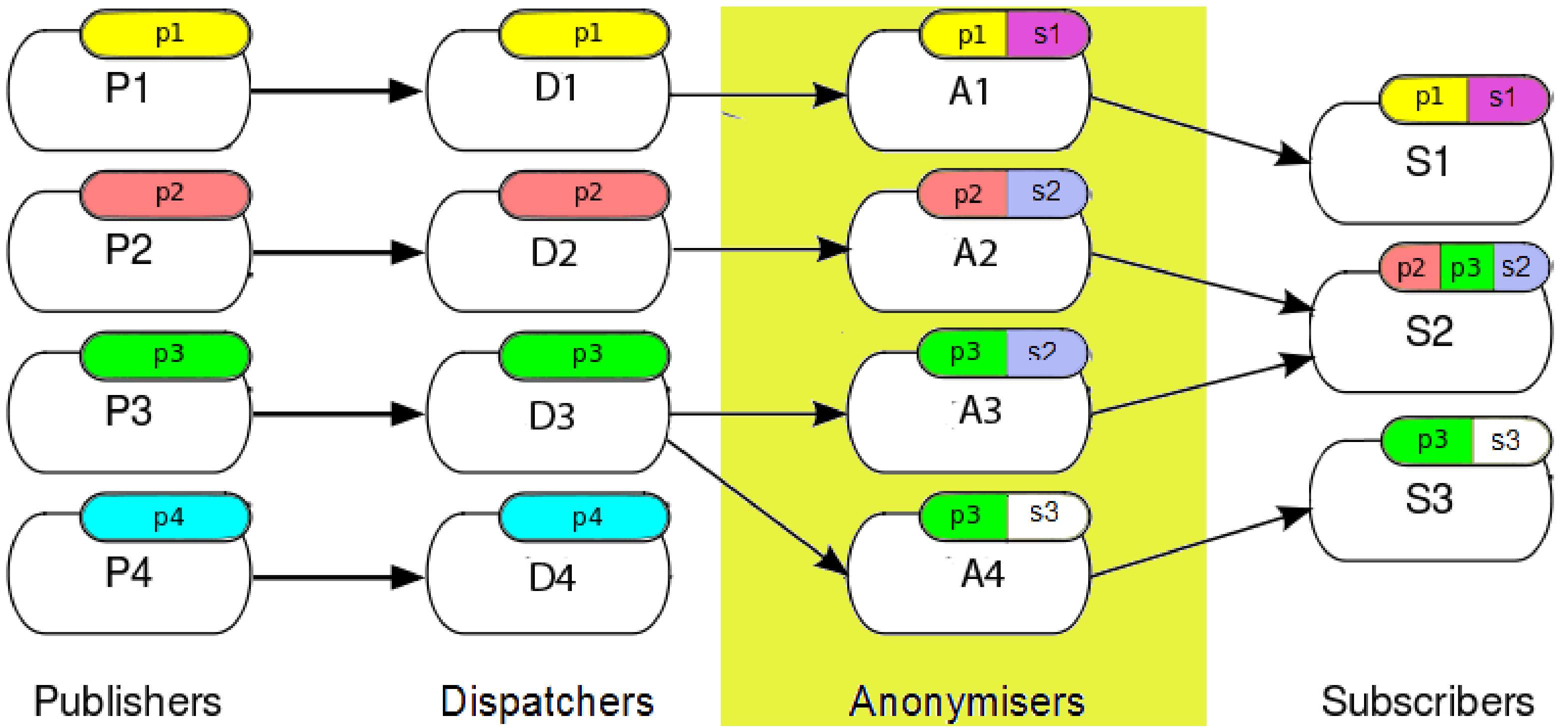


(untrusted code)

Imperial College London

Erlang for Microblogging II

2. *Authorised subscribers shall not be disclosed to any other publisher or subscriber.*

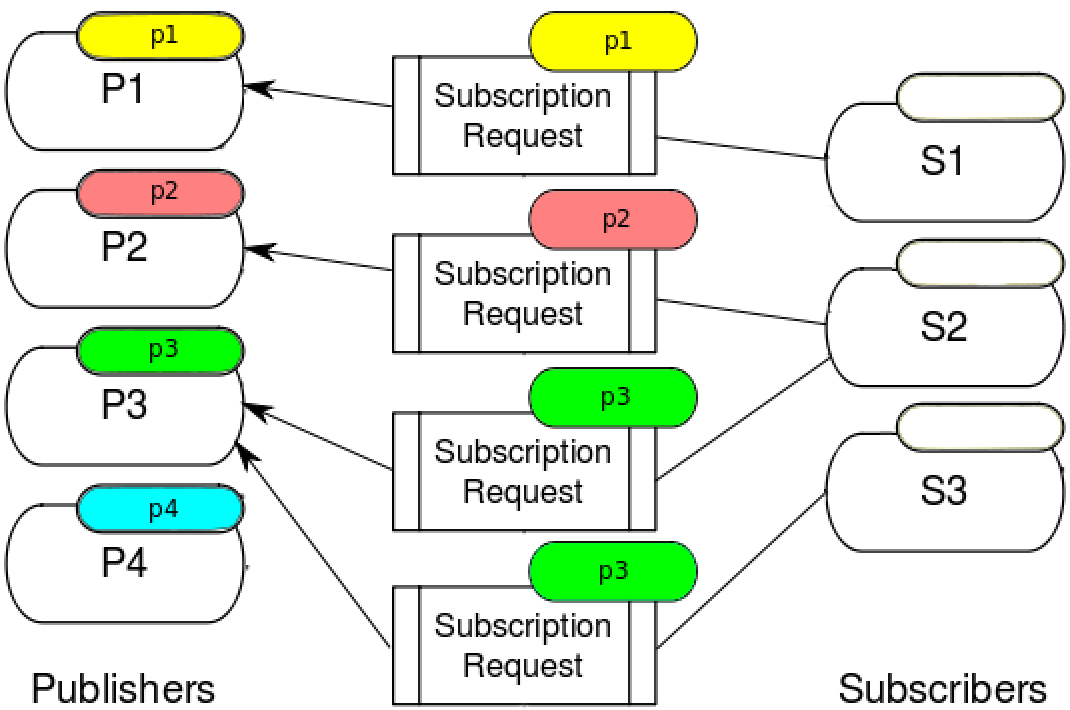


(bug prevention)

Imperial College London

Erlang for Microblogging III

3. *Subscription authorisation requests from subscribers shall be delivered only to the relevant publisher.*



(bug prevention)

Experimental Setup

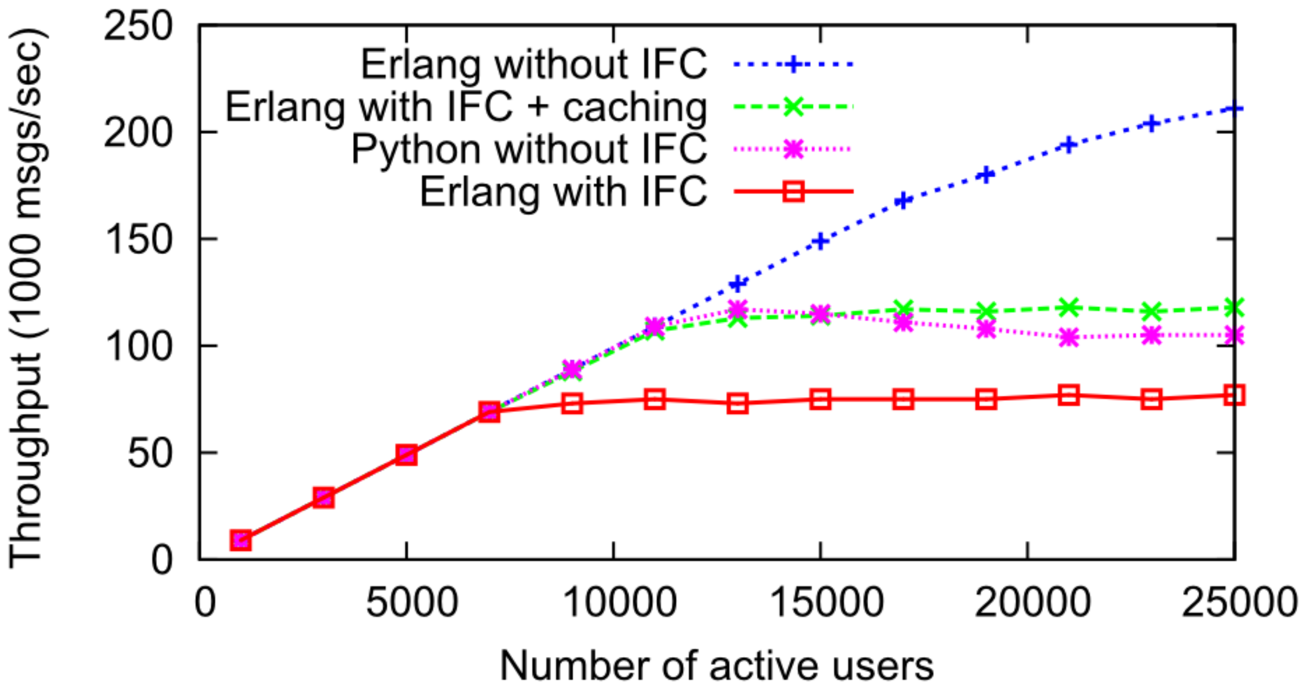
- Erlang Library that provides the IFC API
- Measure throughput in terms of messages per second
- #publishers=#subscribers, 10 subscriptions/subscriber
- Ignored orthogonal issues like message persistence

Comparison between:

- *Python*
  [represents scripting languages]
- *Erlang (no IFC)*
  [Dispatcher per publisher, better multicore performance]
- *Erlang (IFC)*
  [Anonymisers plus label checks]
- *Erlang (IFC + caching)*
  [cache and reuse of label checks]

Imperial College London

Evaluation



Imperial College London

Limitations & Discussion

- **Complexity**
  - Applications have to handle tags/privileges manually
  - Deciding upon a tag allocation scheme is challenging
  - Handling tags routines must be correct for secure operation
  - Policy languages may come to the rescue

- **Persistence**
  - Messages must be stored permanently
  - Fetching and storing data but be compatible with labels
  - Extend Mnesia to be label aware

- **Scalability**
  - Inactive users must be offloaded from RAM
  - Scalability depends upon the ability to keep in memory only the required state
  - Introduce a primitive to hibernate/restore a process

Conclusion

Erlang is an attractive approach for web applications that use IFC to provide privacy guarantees:

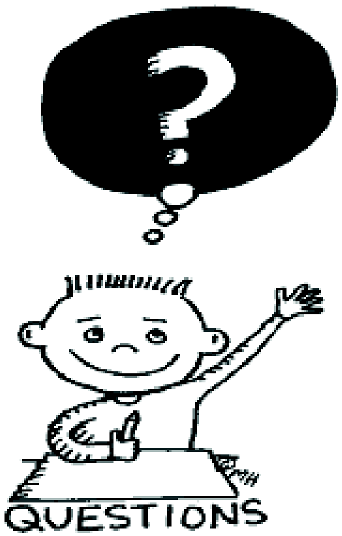
- Isolation of components is free
- Asynchronous message passing is the norm in IFC systems
- Scales well in multicore architectures

Web applications can provide IFC-enabled Erlang APIs and hosting facilities for untrusted extensions

- The web application has to disseminate tags to components according to the relationships between users
- Tags can enforce that the third-party extensions do not violate high level policy

Imperial College London

The End



Ioannis Papagiannis

DoC, Imperial College London

ip108@doc.ic.ac.uk

Imperial College London

Related Work

[How are Erlang Processes Lightweight? 2006]

- Stack frames can be resized/moved (mem)
- User-level, efficient caching when switching (time)
- Lack of shared state means no locking (time)

[xBook09]

- Uses a subset of JavaScript on the server side
- Recreates Erlang’s communication model

[Abestos05]

- Lightweight OS Processes, one per user
- Cooperative Scheduling