Shared Session Types for Safe, Practical Concurrency[1]

Stephanie Balzer
Carnegie Mellon University

Typelevel Summit Philadelphia 2019

1 Supported by a Mozilla Research Grant and NSF Grant No. CCF-1718267

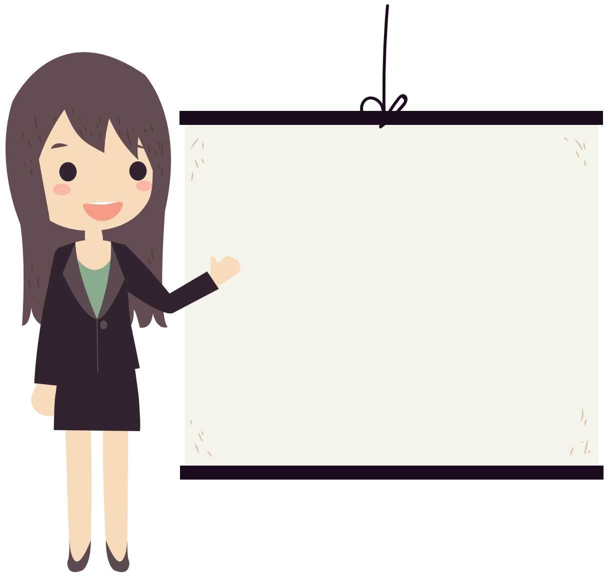
Concurrency is ubiquitous

Concurrency is ubiquitous

The world surrounding us is inherently concurrent

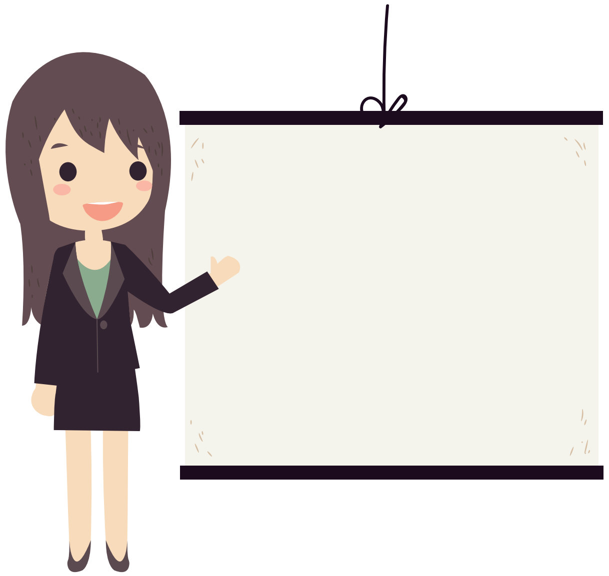
Concurrency is ubiquitous

The world surrounding us is inherently concurrent



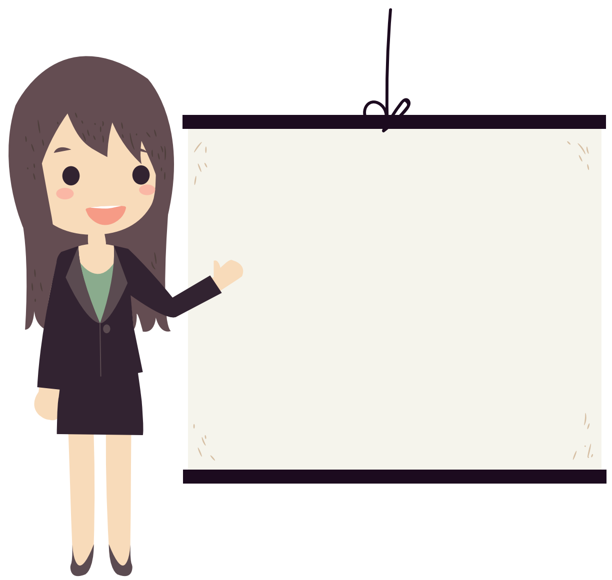
Concurrency is ubiquitous

The world surrounding us is inherently concurrent



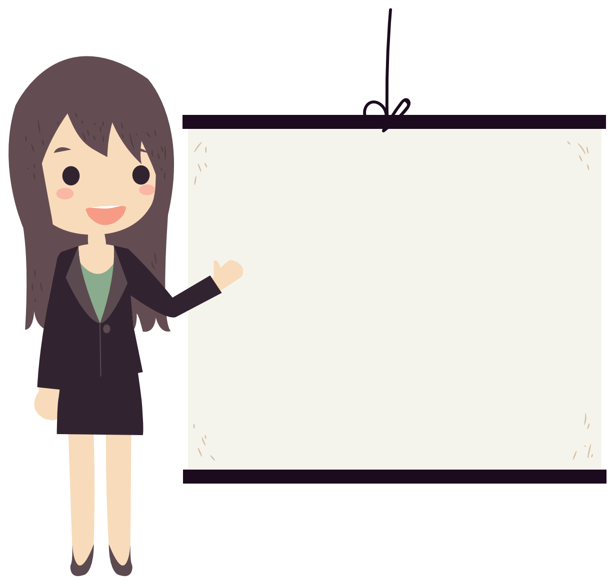
Concurrency is ubiquitous

The world surrounding us is inherently concurrent



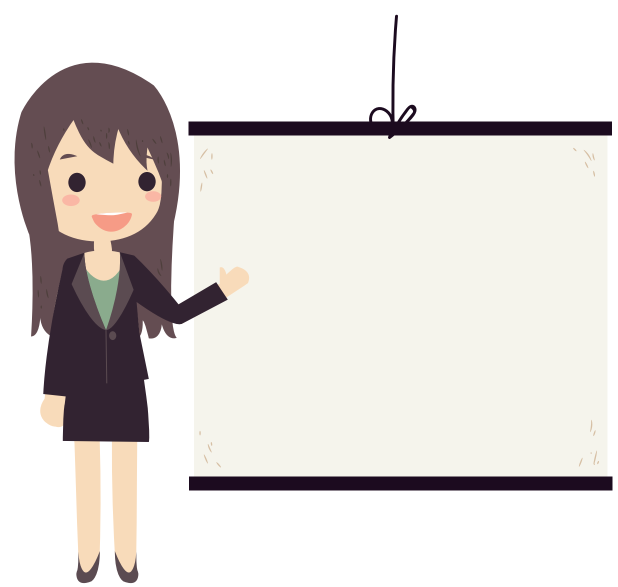
Concurrency is ubiquitous

The world surrounding us is inherently concurrent



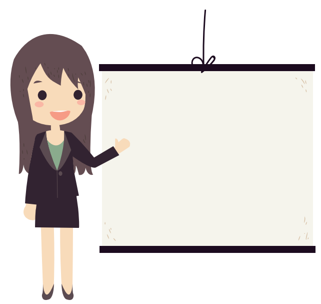
Concurrency is ubiquitous

The world surrounding us is inherently concurrent

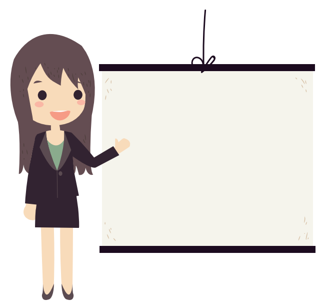
Concurrency is ubiquitous

The world surrounding us is inherently concurrent



Concurrency is ubiquitous

The world surrounding us is inherently concurrent

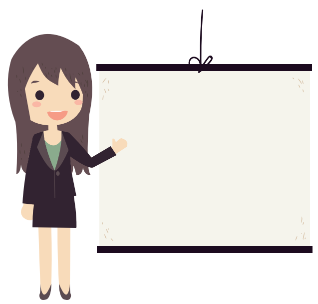


Many programming problems demand concurrency

- Flight booking system, online store, search engines, etc.

Concurrency is ubiquitous

The world surrounding us is inherently concurrent



Many programming problems demand concurrency

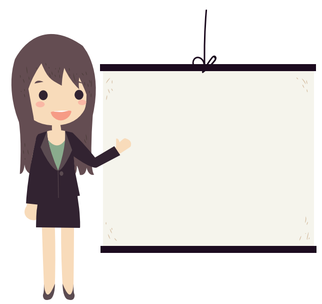
- Flight booking system, online store, search engines, etc.

Computing devices themselves are concurrent

- Run various apps concurrently

Concurrency is ubiquitous

The world surrounding us is inherently concurrent



Many programming problems demand concurrency

- Flight booking system, online store, search engines, etc.

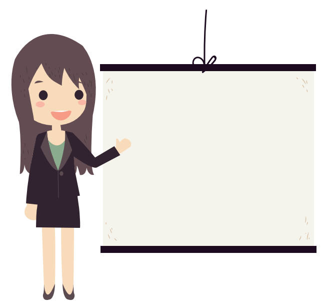
Computing devices themselves are concurrent

- Run various apps concurrently

→ **programming languages must support concurrency**

Concurrency is ubiquitous

The world surrounding us is inherently concurrent



Many programming problems demand concurrency

- Flight booking system, online store, search engines, etc.

Computing devices themselves are concurrent

- Run various apps concurrently

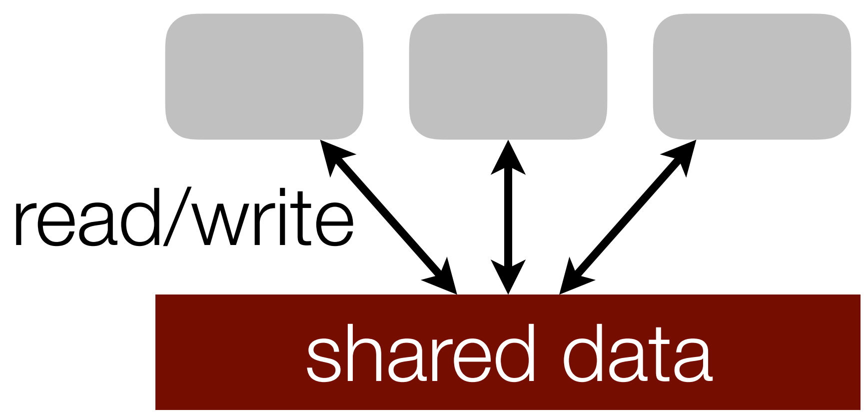
→ programming languages must support concurrency

→ concurrent programming is notoriously difficult and error-prone

Two models for concurrent programming

Two models for concurrent programming

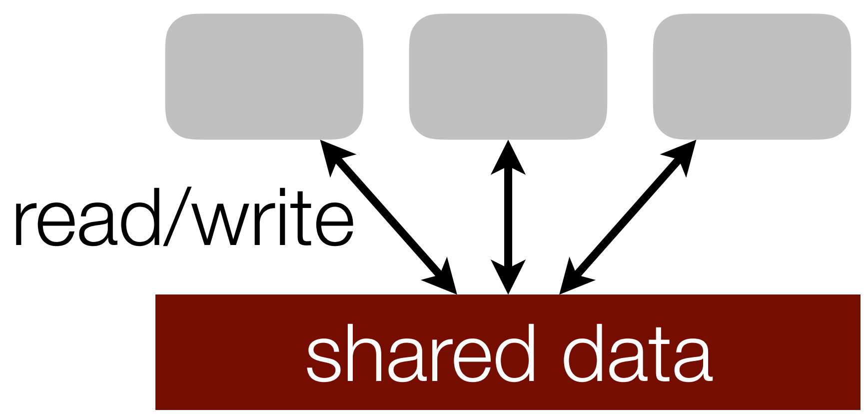
Shared memory



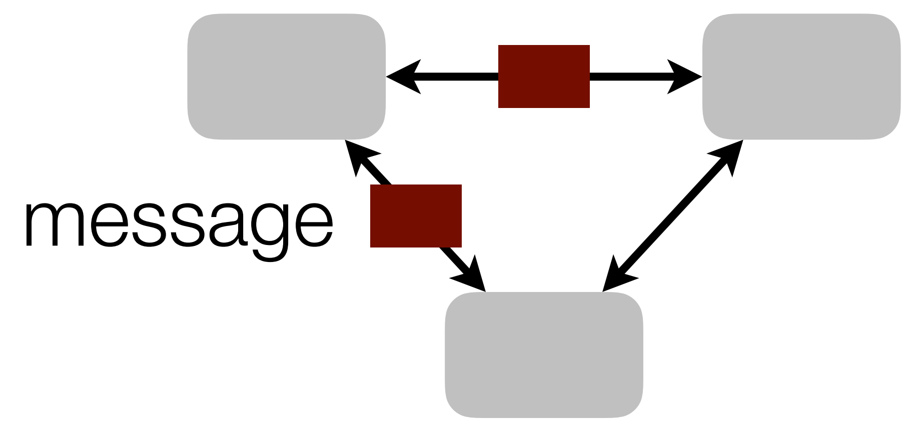
Legend: concurrently executing component

Two models for concurrent programming

Shared memory



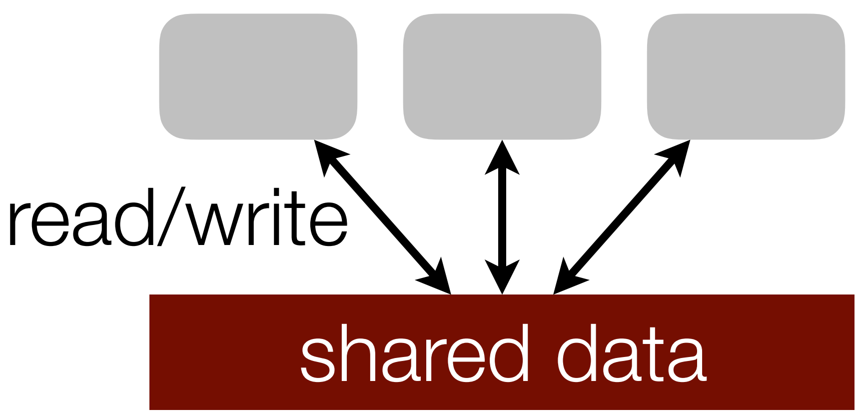
Message-passing



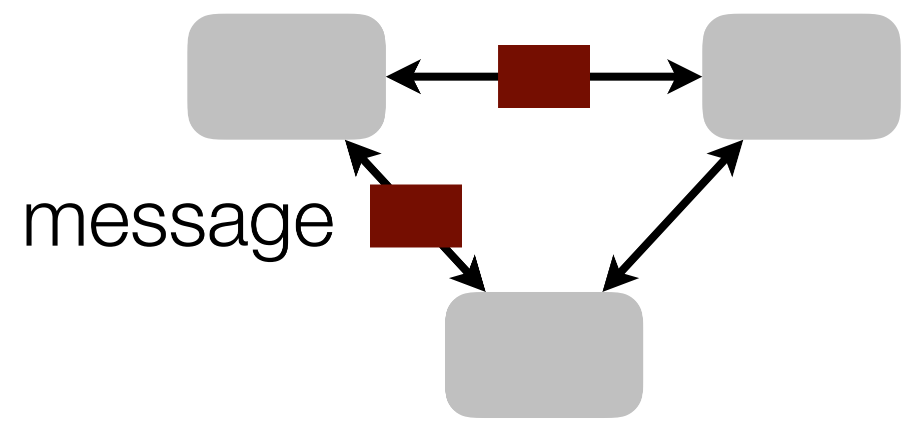
Legend: concurrently executing component

Two models for concurrent programming

Shared memory



Message-passing

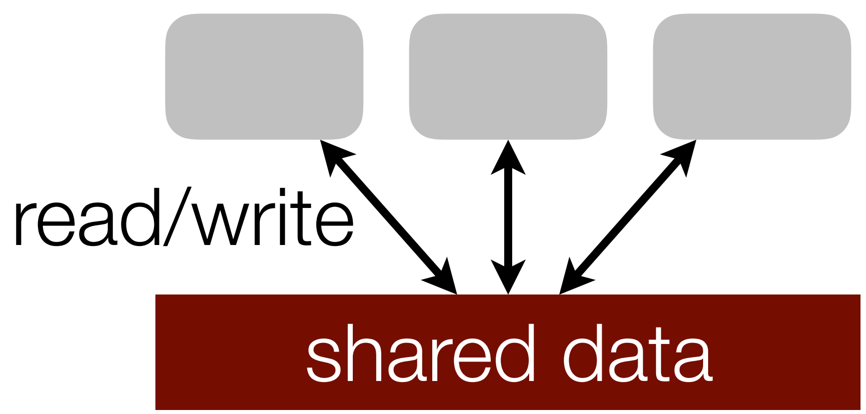


- computation by reading from and writing to **shared data**

Legend:  concurrently executing component

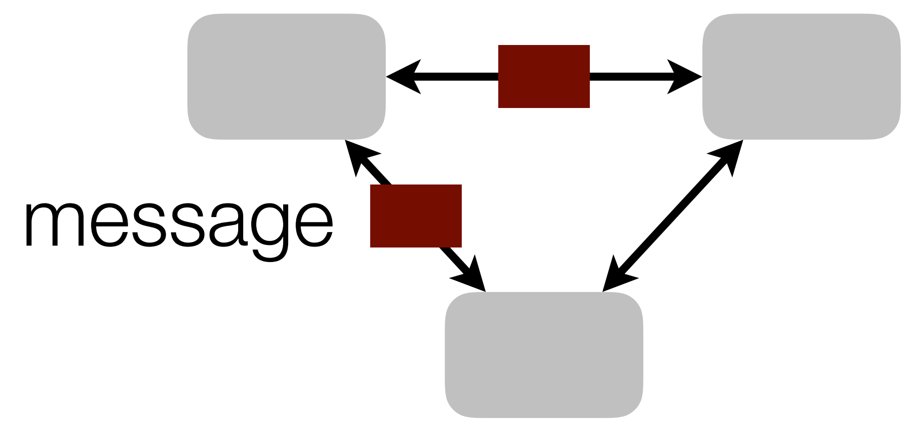
Two models for concurrent programming

Shared memory



- computation by reading from and writing to **shared data**

Message-passing

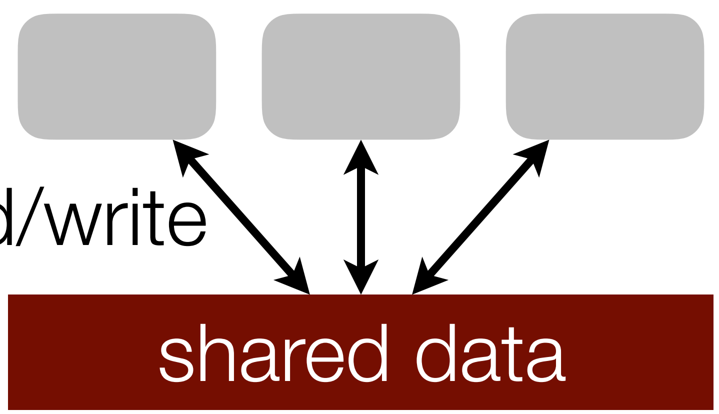


- computation by exchange of **messages**

Legend: concurrently executing component

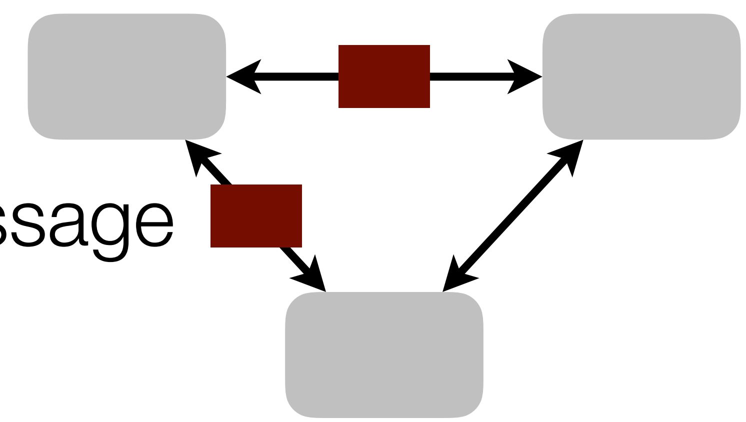
Two models for concurrent programming

Shared memory



- computation by reading from and writing to **shared data**

Message-passing



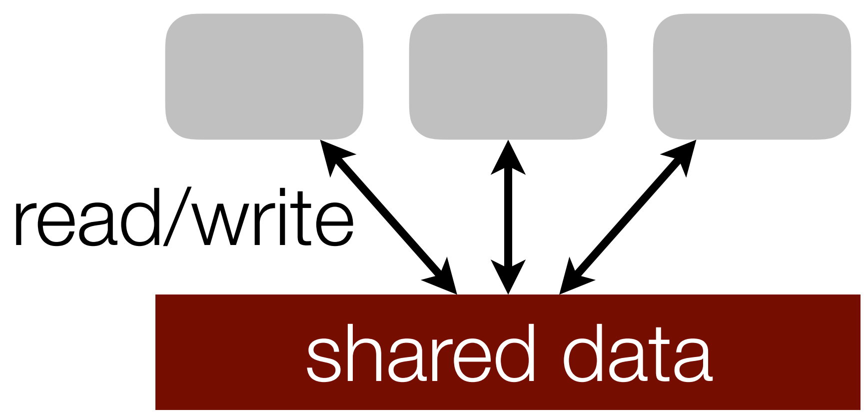
- computation by exchange of **messages**

→ message-passing offers higher-level of abstraction

Legend: [grey box] concurrently executing component

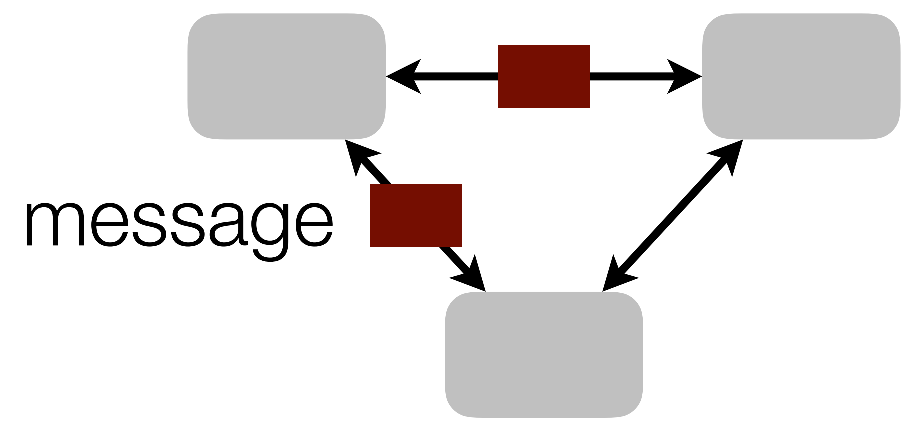
Two models for concurrent programming

Shared memory



- computation by reading from and writing to **shared data**

Message-passing



- computation by exchange of **messages**

message-passing offers higher-level of abstraction

message-passing adopted by practical languages such as Erlang, Go, and Rust.

Legend: concurrently executing component

My research

My research

Goal: make concurrent programming safe and practical

My research

Goal: make concurrent programming safe and practical

- message-passing model
- session types to express protocols of message exchange and reason sequentially about communicating parties

My research

Goal: make concurrent programming safe and practical

- message-passing model
- session types to express protocols of message exchange and reason sequentially about communicating parties

Contributions:

My research

Goal: make concurrent programming safe and practical

- message-passing model
- session types to express protocols of message exchange and reason sequentially about communicating parties

Contributions:

- shared session types

My research

Goal: make concurrent programming safe and practical

- message-passing model
- session types to express protocols of message exchange and reason sequentially about communicating parties

Contributions:

- shared session types
  - accommodate real-world programming scenarios
  - guarantee protocol adherence, data-race-freedom, and deadlock-freedom

Session types, what are they? Why do we need them in practice?

Message-passing concurrency in Servo

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

  Image loader:

Message-passing concurrency in Servo

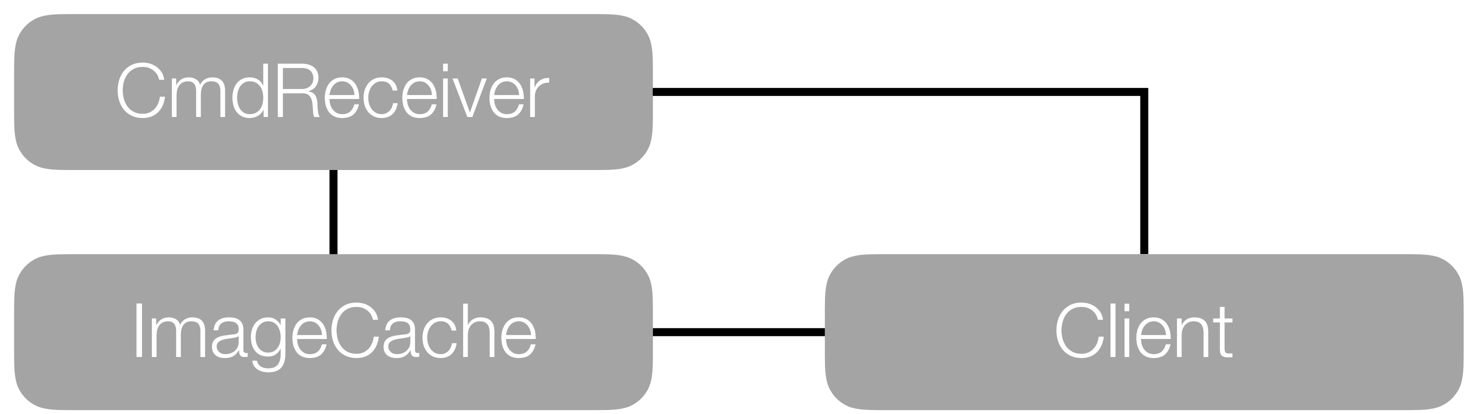
- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

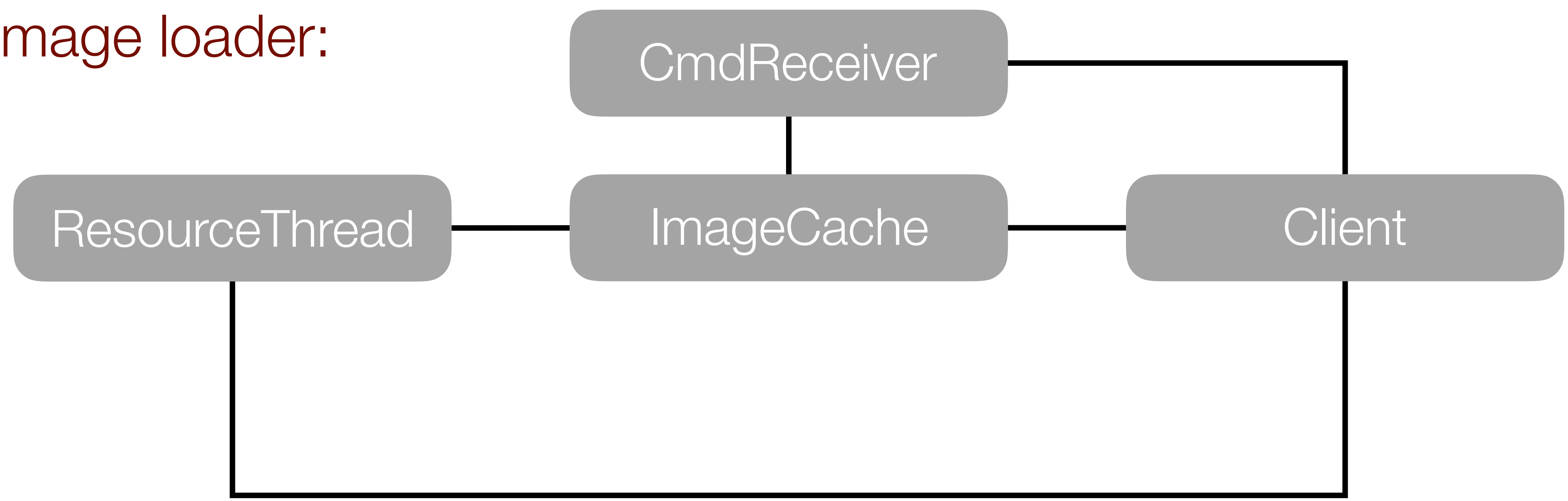


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

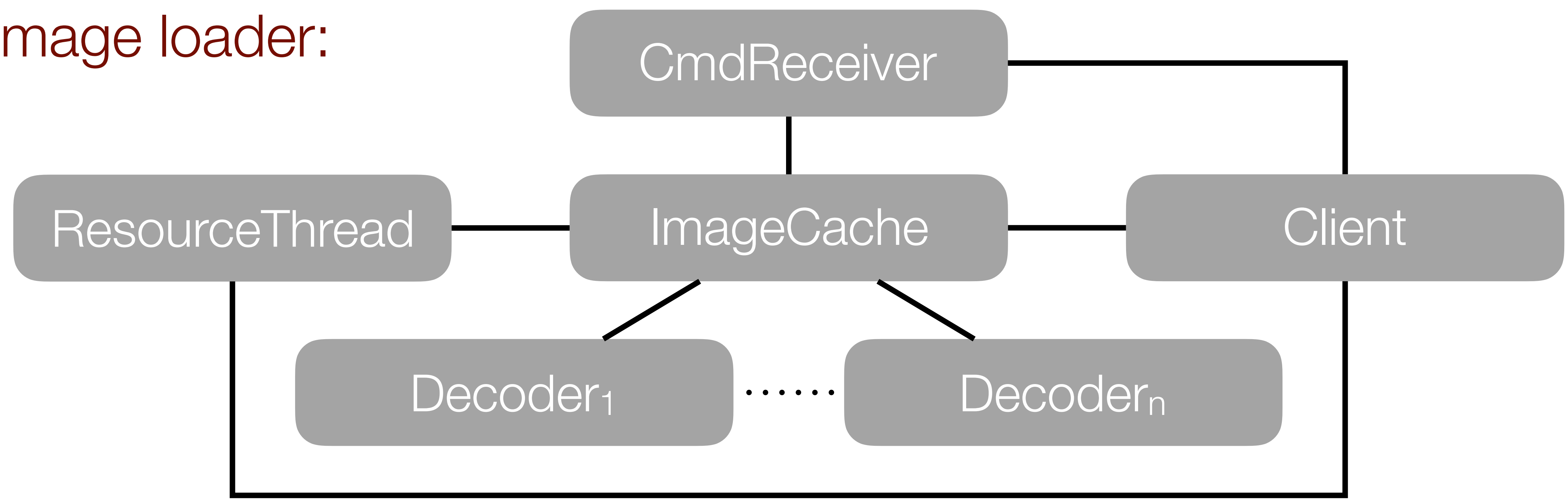


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

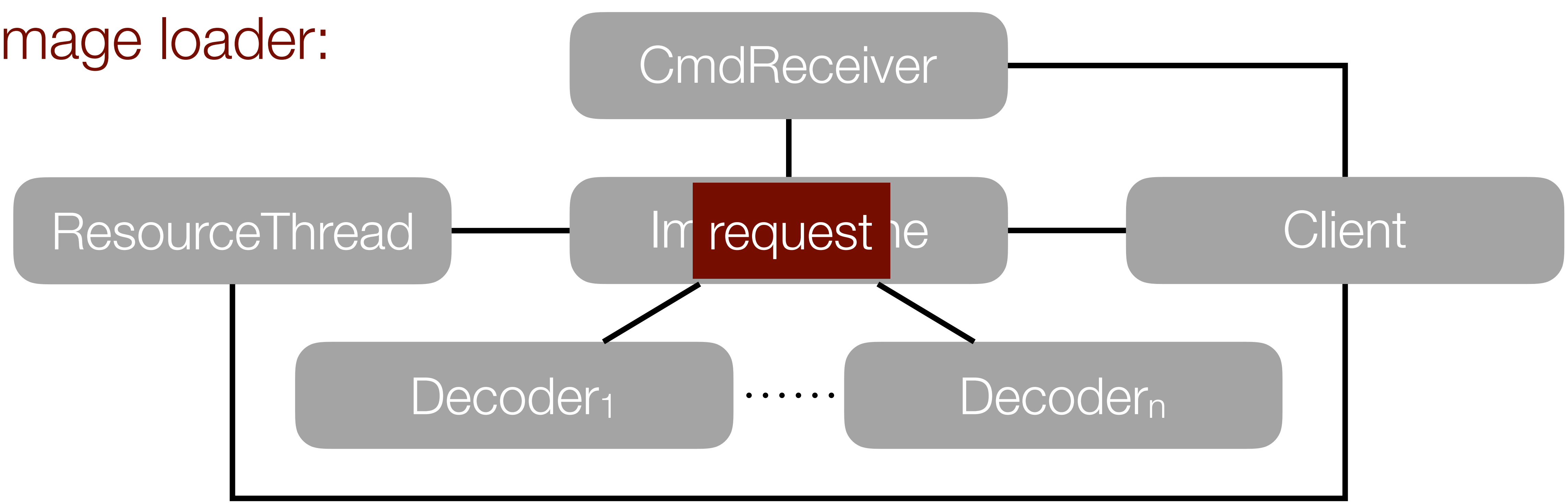


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

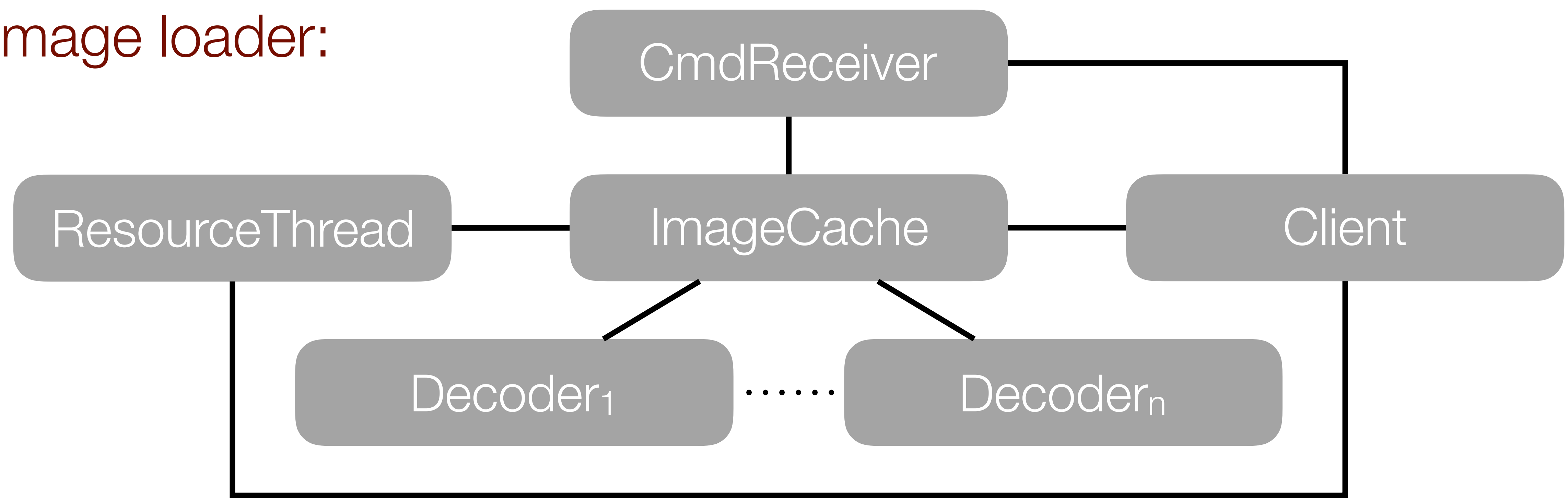


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

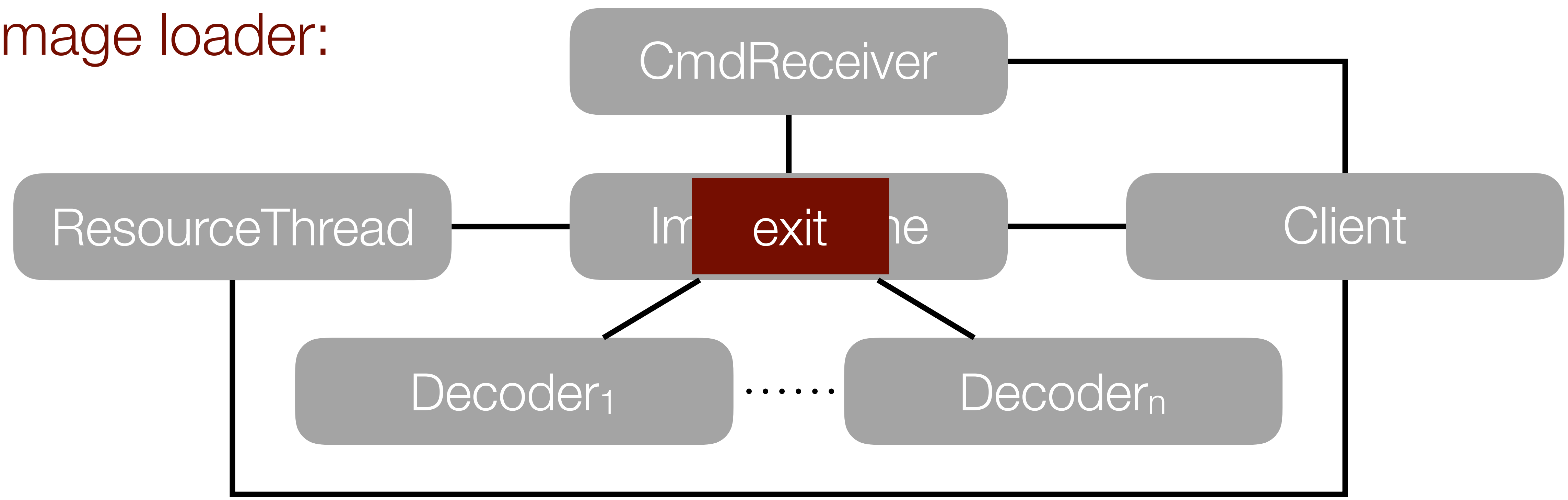


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

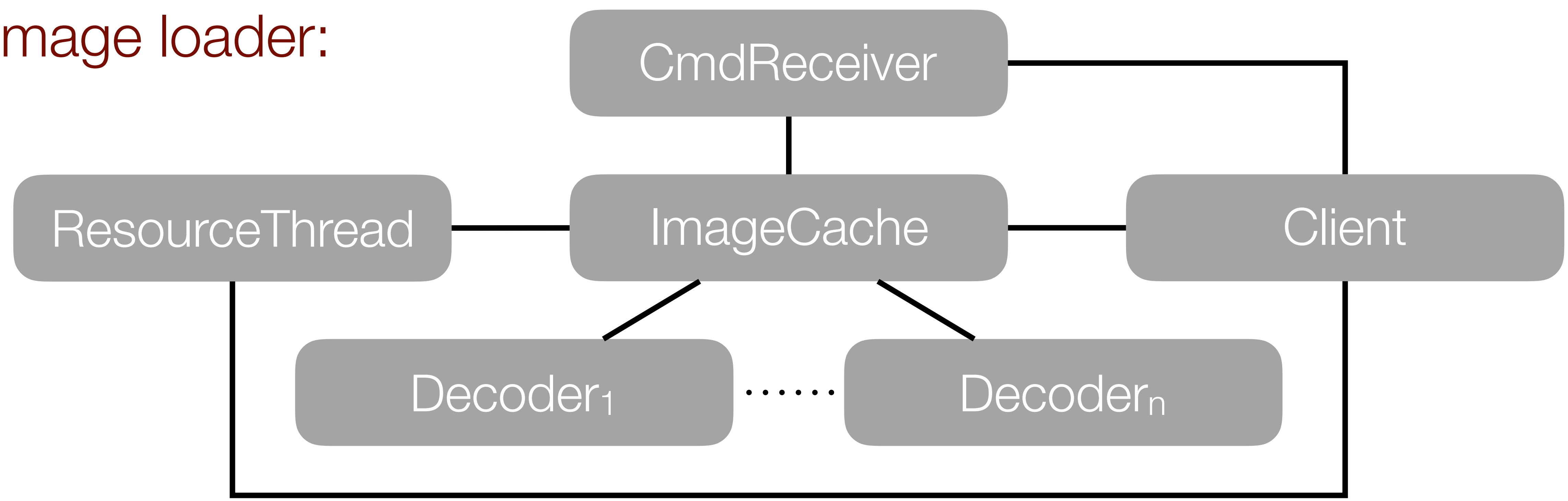


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

Image loader:

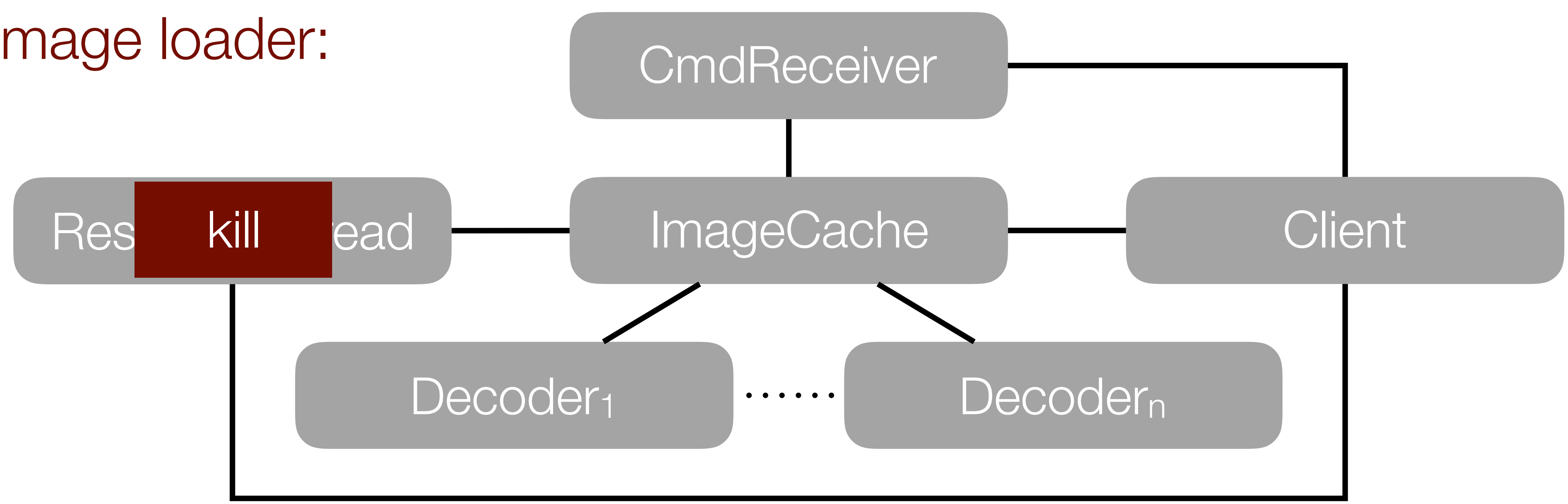


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

- Servo is Mozilla's next-generation browser engine under development and implemented in Rust.
- Servo uses message-passing concurrency for maximal parallelization of tasks, such as loading and rendering of webpage elements.

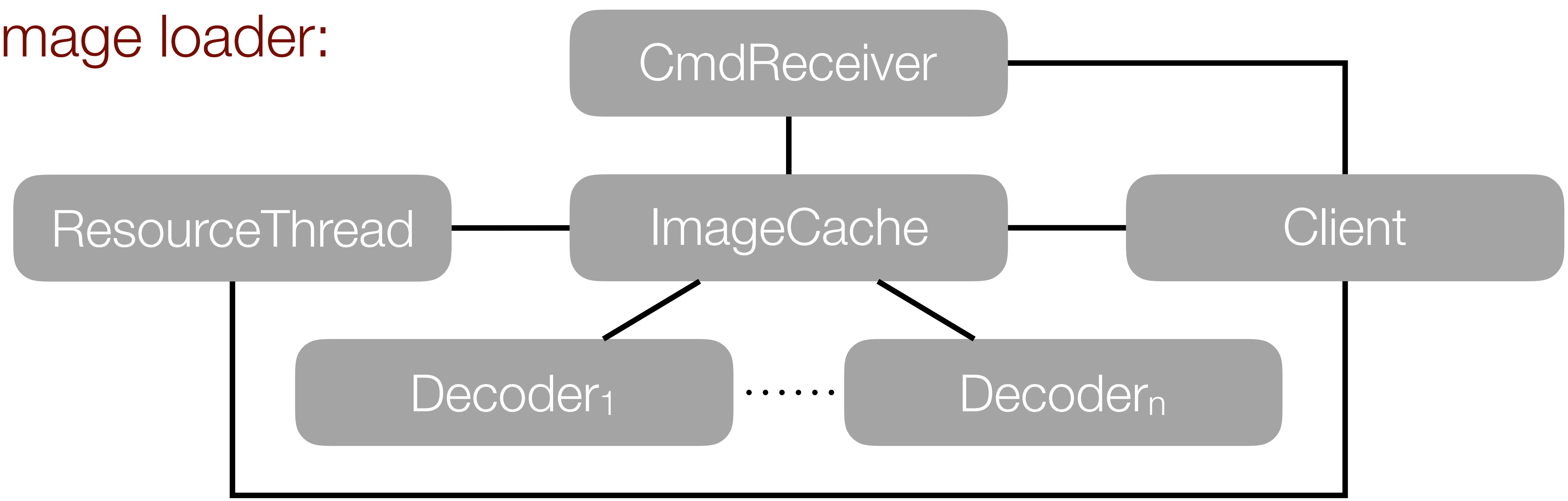
Image loader:



Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

Image loader:

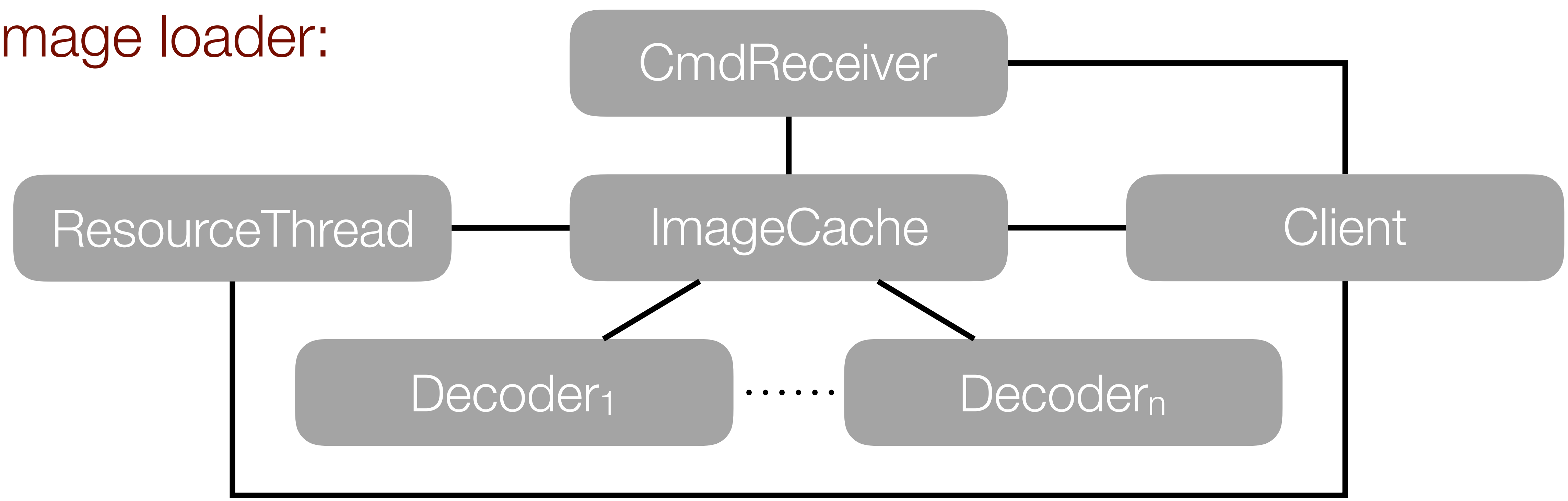


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

To restrict the kinds of messages that can be sent over a channel, Rust channels are typed with enumeration types.

Image loader:

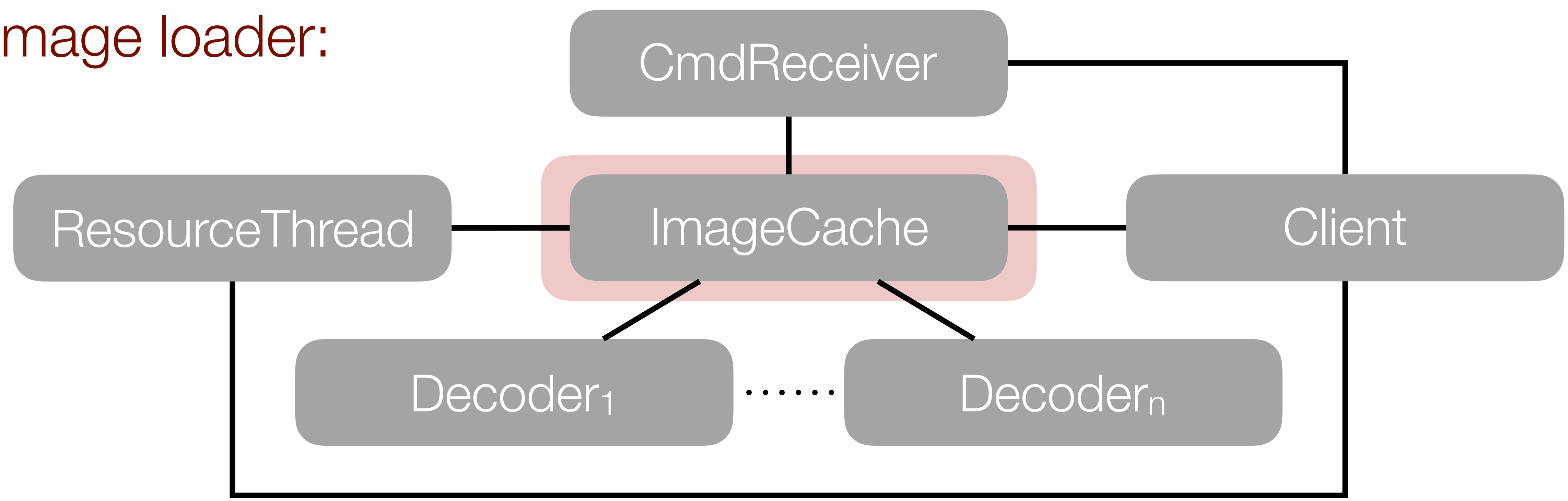


Legend: component, runs in separate thread — channel

Message-passing concurrency in Servo

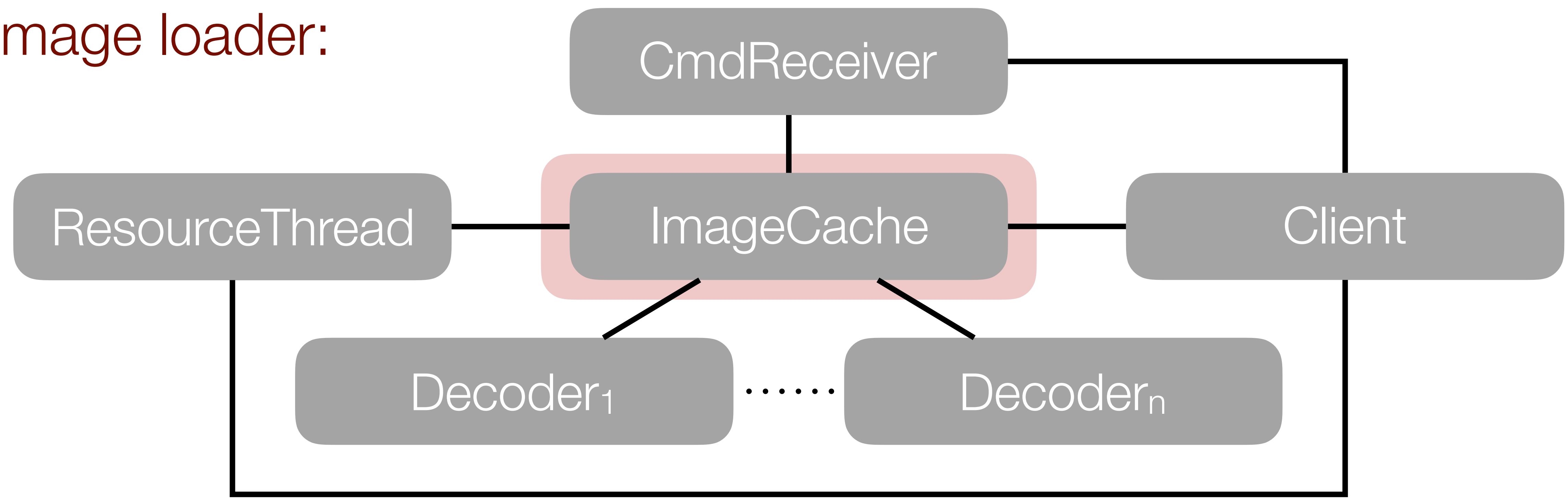
- To restrict the kinds of messages that can be sent over a channel, Rust channels are typed with enumeration types.
- Example: enumeration for ImageCache

Image loader:



Legend: component, runs in separate thread — channel

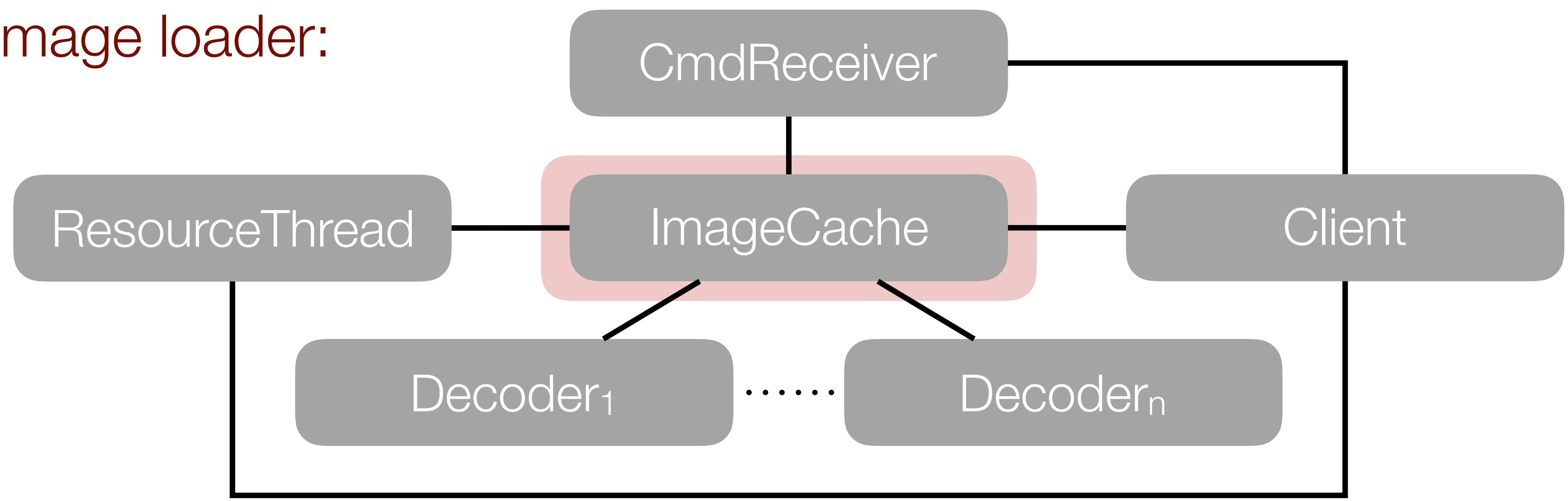
Image loader:



Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
  GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

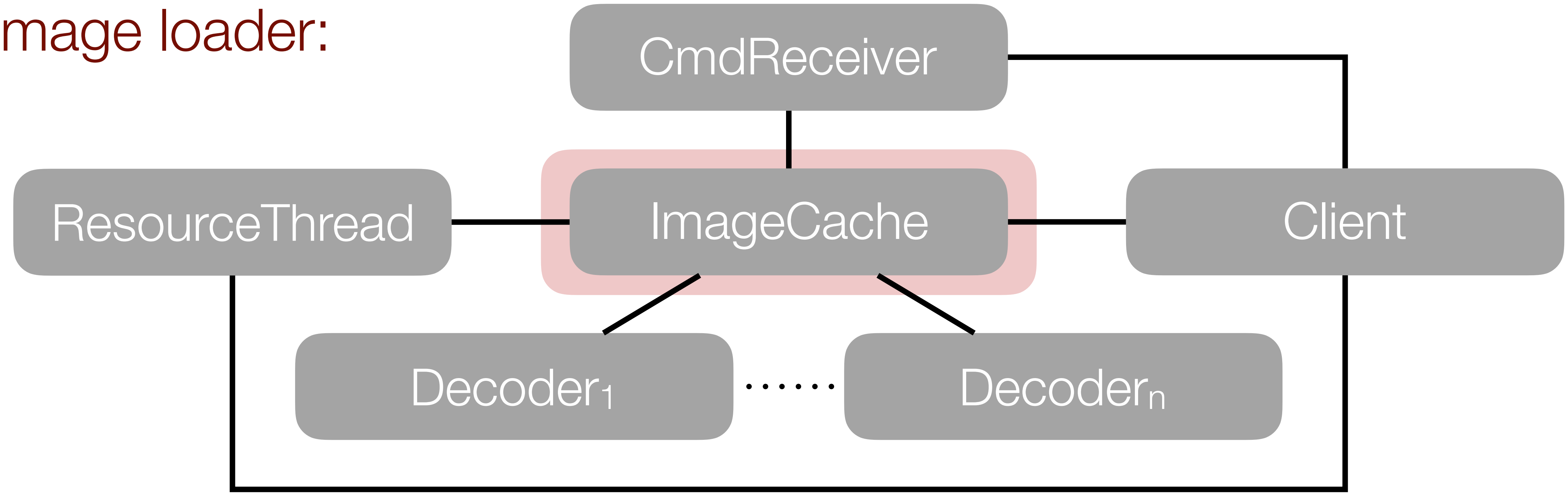
Image loader:



Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
  GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

Image loader:

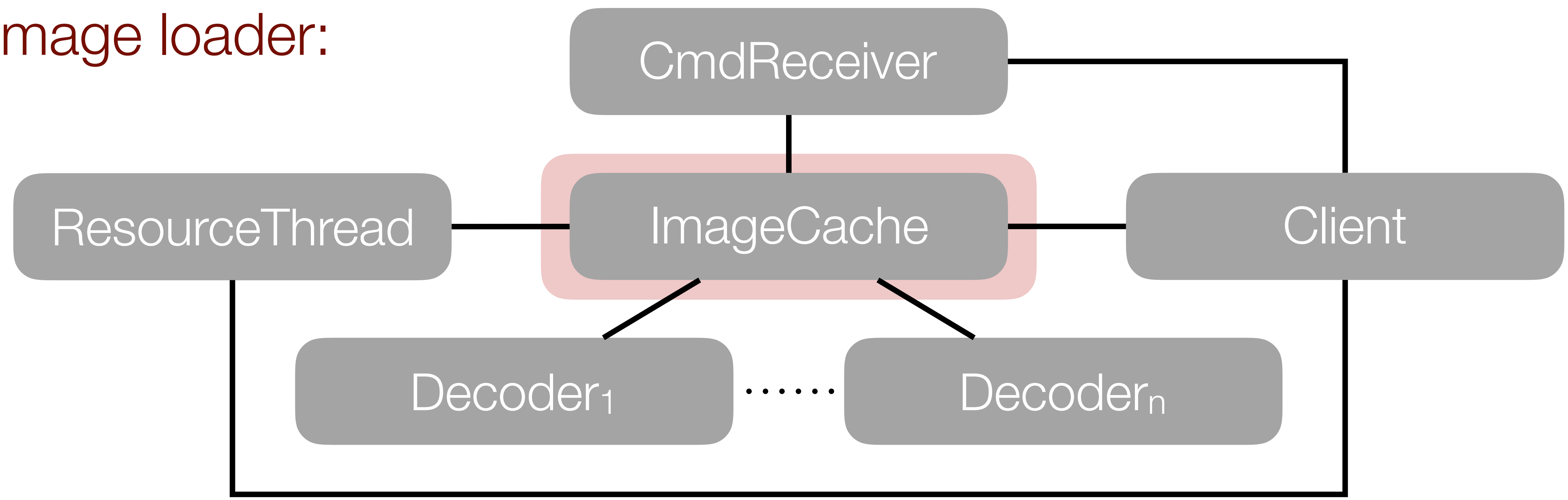


Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
  GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down
  Exit ()
}
```

one variant for each message

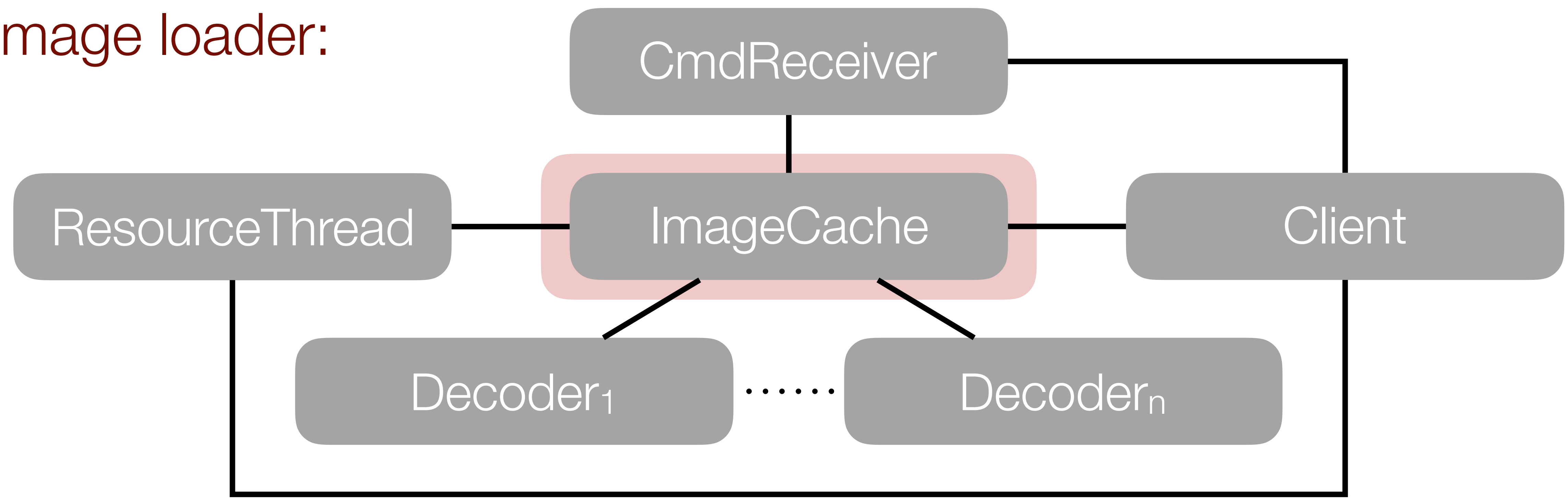
Image loader:



Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
  GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

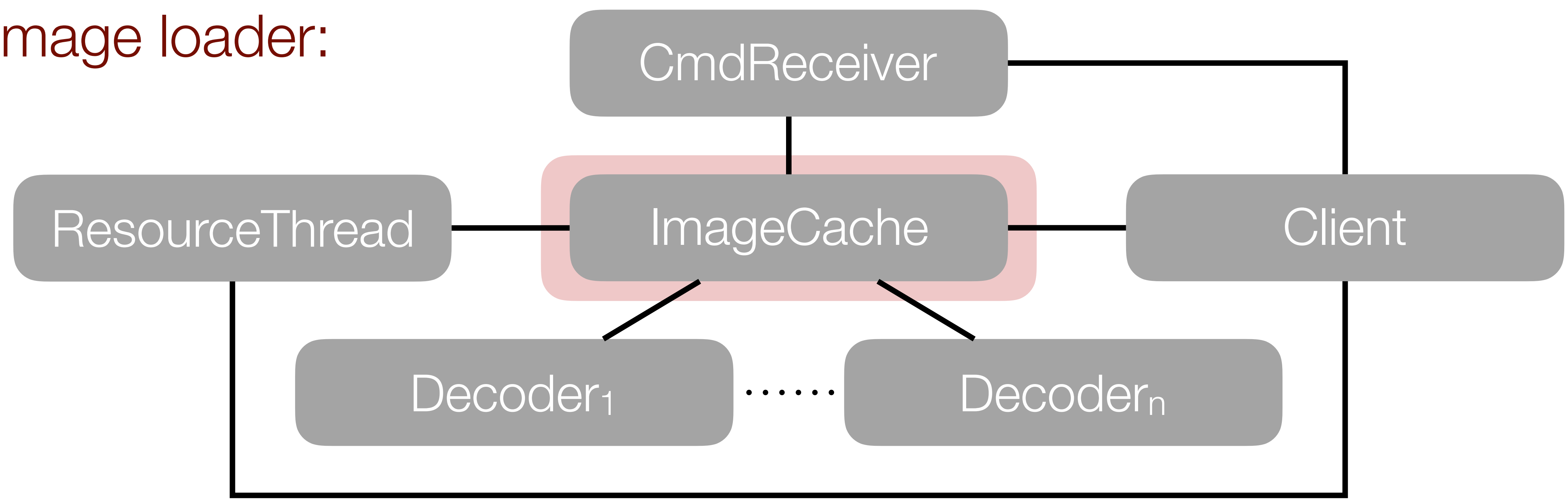
Image loader:



Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
  GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

Image loader:

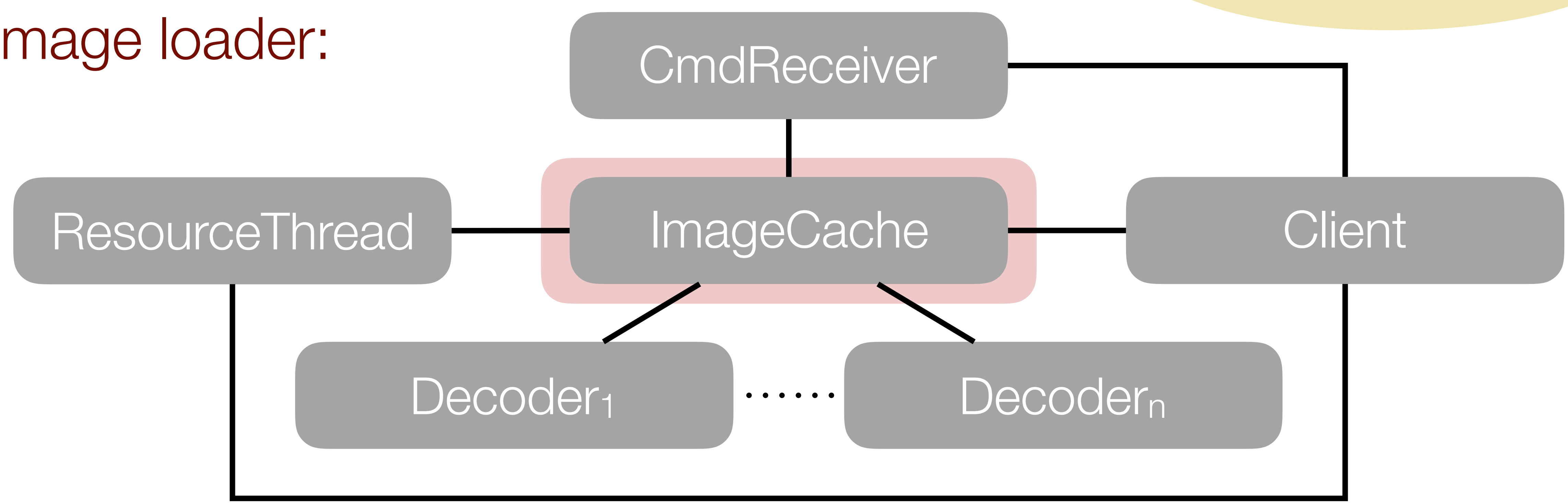


Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
  GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

implicit protocol

Image loader:



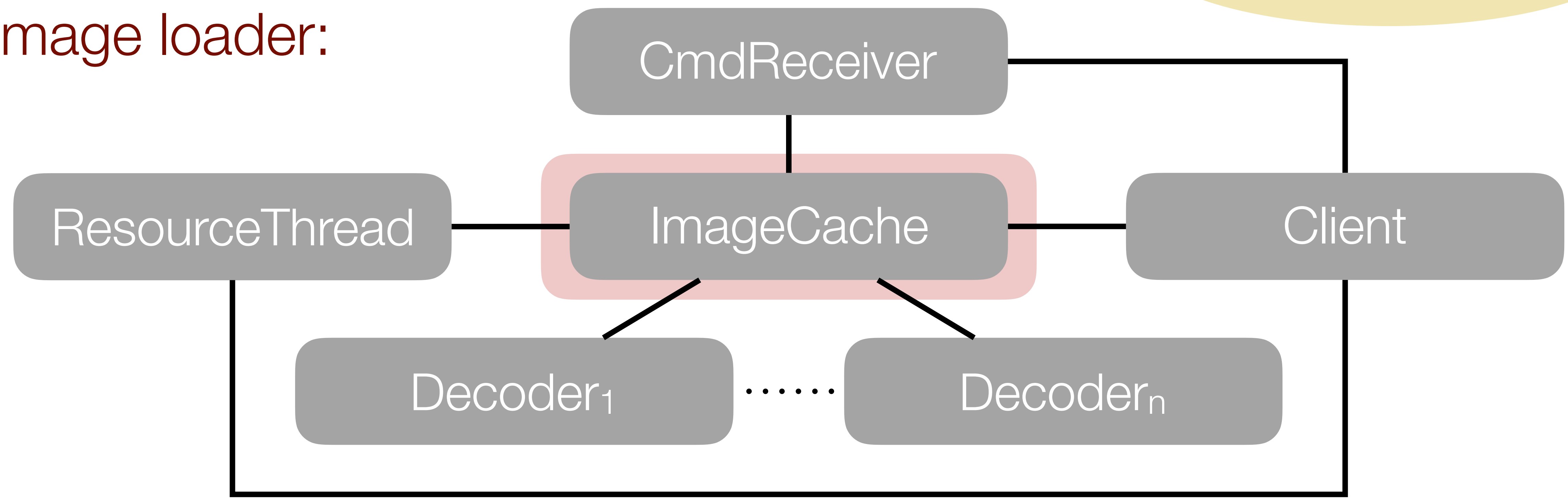
Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Opt…
  GetImageIfAvailable (Url, UsePlaceholder, Ip…                                        >),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

protocol breaches
result in proliferation of panic! and
infinite waiting

implicit protocol

Image loader:



Legend: component, runs in separate thread — channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Opt
  GetImageIfAvailable (Url, UsePlaceholder, Ip                              >),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

protocol breaches
result in proliferation of panic! and
infinite waiting

implicit protocol

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Opt
  GetImageIfAvailable (Url, UsePlaceholder, Ipc                                   >),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

protocol breaches
result in proliferation of panic! and
infinite waiting

implicit protocol

enumeration types ensure that only defined messages can be communicated along a channel

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Opt…
  GetImageIfAvailable (Url, UsePlaceholder, Ip… >),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

protocol breaches result in proliferation of panic! and infinite waiting

implicit protocol

➡ enumeration types ensure that only defined messages can be communicated along a channel

➡ enumeration types fail to ensure that messages are sent according to the intended protocol

```
pub enum ImageCacheCommand {
  RequestImage (Url, ImageCacheChan, Opti
  GetImageIfAvailable (Url, UsePlaceholder, Ipc                                    >),
  StoreDecodeImage (Url, Vec<u8>),
  ...
  // Clients must wait for a response before shutting down ResourceThread
  Exit ()
}
```

protocol breaches
result in proliferation of panic! and
infinite waiting

implicit protocol

- enumeration types ensure that only defined messages can be communicated along a channel
- enumeration types fail to ensure that messages are sent according to the intended protocol
- let’s use session types!

Session types

Session types

Session types define protocols of message exchange.

Session types

Session types define protocols of message exchange.

“protocol = sequence of actions”

Session types

Session types define protocols of message exchange.

“protocol = sequence of actions”

$$
\begin{array}{llll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

Session types

- Session types define protocols of message exchange.
- "protocol = sequence of actions"

A, B	$\triangleq$	$\&\{\overline{l_i : A_i}\}$	external choice
		$\oplus\{\overline{l_i : A_i}\}$	internal choice
		$A \multimap B$	channel input
		$A \otimes B$	channel output
		$\mathbf{1}$	termination

client
chooses among sending
one of the labels l_i

Session types

Session types define protocols of message exchange.

“protocol = sequence of actions”

$$
\begin{array}{llll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

Session types

- Session types define protocols of message exchange.
- "protocol = sequence of actions"

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

provider
chooses among sending
one of the labels l_i

Session types

- Session types define protocols of message exchange.
- "protocol = sequence of actions"

$$
\begin{array}{llll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

Session types

Session types define protocols of message exchange.

“protocol = sequence of actions”

A, B	$\triangleq$	$\&\{\overline{l_i : A_i}\}$	external choice
		$\oplus\{\overline{l_i : A_i}\}$	internal choice
		$A \multimap B$	channel input
		$A \otimes B$	channel output
		$\mathbf{1}$	termination

client
sends channel reference of
type A

Session types

- Session types define protocols of message exchange.
- "protocol = sequence of actions"

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

Session types

Session types define protocols of message exchange.

"protocol = sequence of actions"

$$
\begin{array}{llll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

provider
sends channel reference of
type A

Session types

Session types define protocols of message exchange.

"protocol = sequence of actions"

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

Session types

Session types define protocols of message exchange.

"protocol = sequence of actions"

A, B	$\triangleq$	$\&\{\overline{l_i : A_i}\}$	external choice
		$\oplus\{\overline{l_i : A_i}\}$	internal choice
		$A \multimap B$	channel input
		$A \otimes B$	channel output
		$\mathbf{1}$	termination

provider terminates

Session types

Session types define protocols of message exchange.

"protocol = sequence of actions"

A, B	$\triangleq$	$\&\{\overline{l_i : A_i}\}$	external choice
		$\oplus\{\overline{l_i : A_i}\}$	internal choice
		$A \multimap B$	channel input
		$A \otimes B$	channel output
		$\mathbf{1}$	termination

Session types

Session types define protocols of message exchange.

"protocol = sequence of actions"

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \to A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

Session types

- Session types define protocols of message exchange.
- "protocol = sequence of actions"

support communication of values

$$
\begin{array}{llll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \to A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

Session type for image loader

$$
\begin{array}{lll l}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \to A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

Session type for image loader

A, B	$\triangleq$	$\&\{\overline{l_i : A_i}\}$	external choice
		$\oplus\{\overline{l_i : A_i}\}$	internal choice
		$A \multimap B$	channel input
		$A \otimes B$	channel output
		$\mathbf{1}$	termination
		$T \rightarrow A$	value input
		$T \times A$	value output
T	$\triangleq$	$\mathsf{int} \mid \mathsf{string} \mid \ldots$	

$\mathsf{ImgCacheCmd} =$

Session type for image loader

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \rightarrow A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

$\mathsf{ImgCacheCmd} = \& \, \{$

$\}$

Session type for image loader

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \to A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{ImgCacheCmd} = \& & \{\mathsf{RequestImage} : \\
 & \quad \ldots \\
 & \quad \mathsf{Exit} : \\
 & \\
 & \}
\end{array}
$$

Session type for image loader

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \to A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\ \{ & \mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\
& \ldots \\
& \mathsf{Exit} : \\
\} &
\end{aligned}
$$

Session type for image loader

$$
\begin{array}{lcll}
A, B & \triangleq & \&\{\overline{l_i : A_i}\} & \text{external choice} \\
 & & \oplus\{\overline{l_i : A_i}\} & \text{internal choice} \\
 & & A \multimap B & \text{channel input} \\
 & & A \otimes B & \text{channel output} \\
 & & \mathbf{1} & \text{termination} \\
 & & T \to A & \text{value input} \\
 & & T \times A & \text{value output} \\
T & \triangleq & \mathsf{int} \mid \mathsf{string} \mid \ldots &
\end{array}
$$

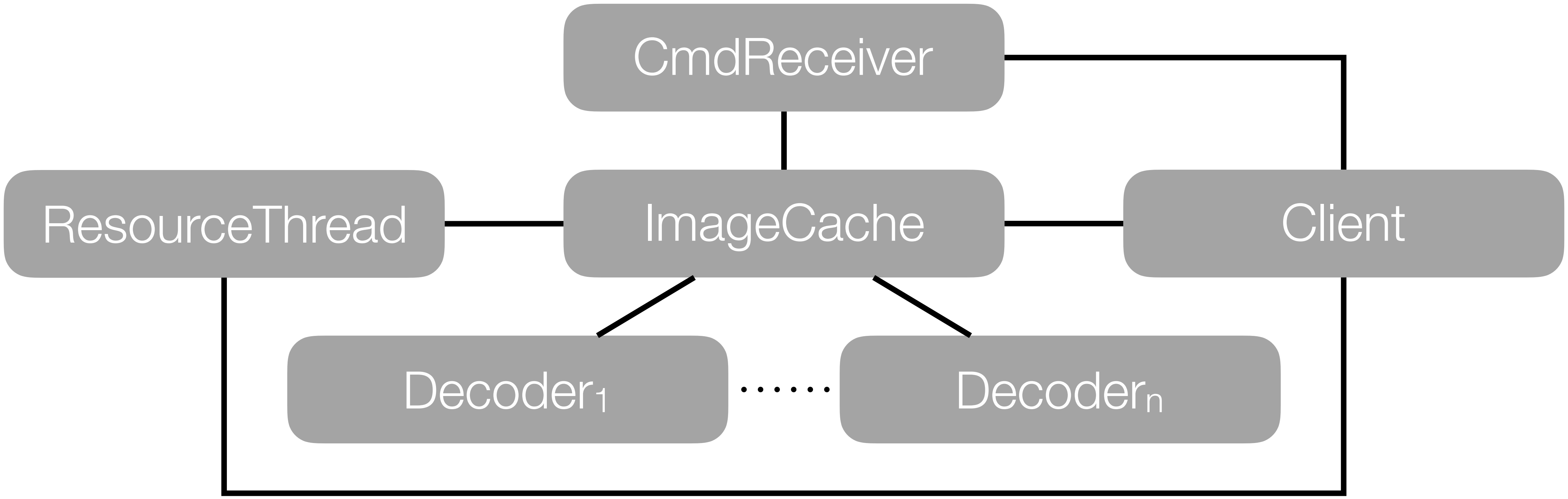
$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \& \{ & \mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\
& \ldots \\
& \mathsf{Exit} : \oplus \{ \mathsf{Running} : \mathsf{ImgCacheCmd}, \\
& \qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1} \} \\
\} &
\end{aligned}
$$

Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\ \{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\ \{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

Session type for image loader

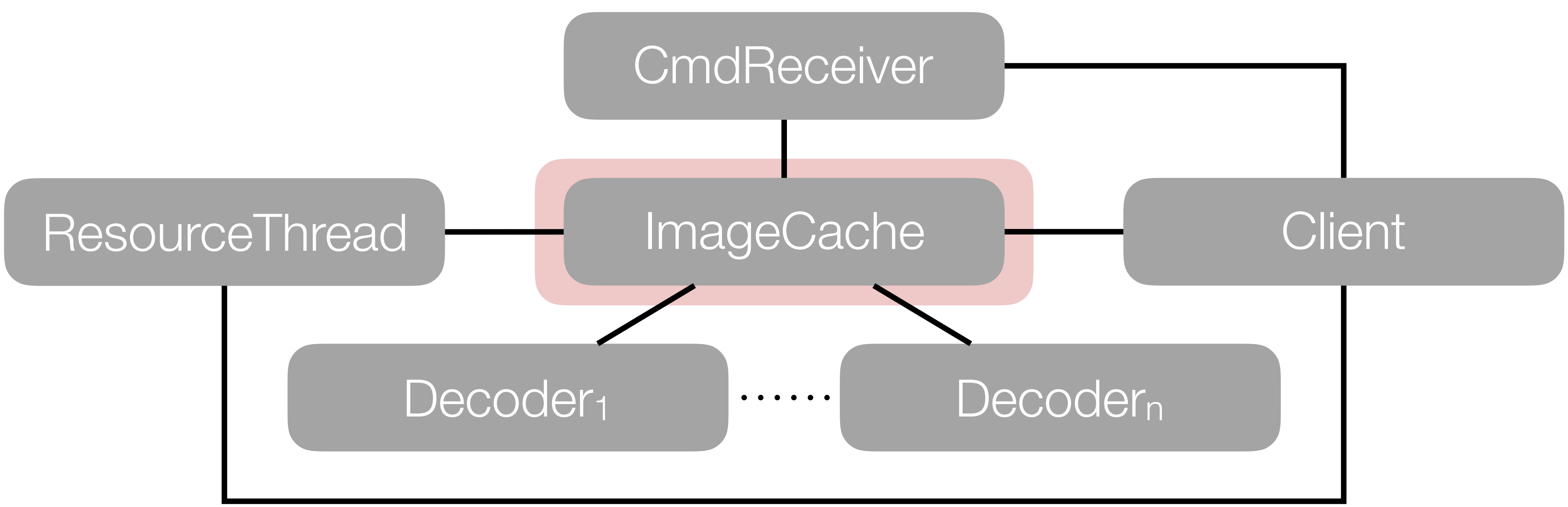
$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \mathbin{\&}\ \{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\
&\ldots\\
&\mathsf{Exit} : \oplus\ \{\mathsf{Running} : \mathsf{ImgCacheCmd},\\
&\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\
\}&
\end{aligned}
$$



Session type for image loader

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\
&\ldots\\
&\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\
&\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\
\}&
\end{aligned}
$$

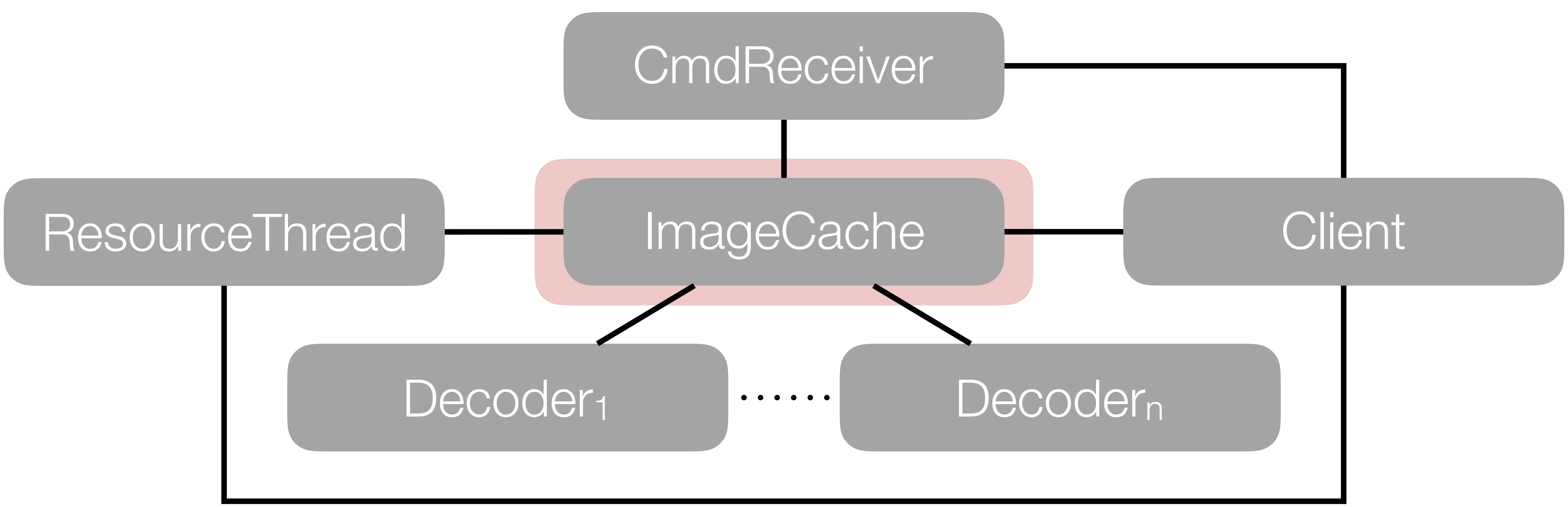
ImageCache:



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

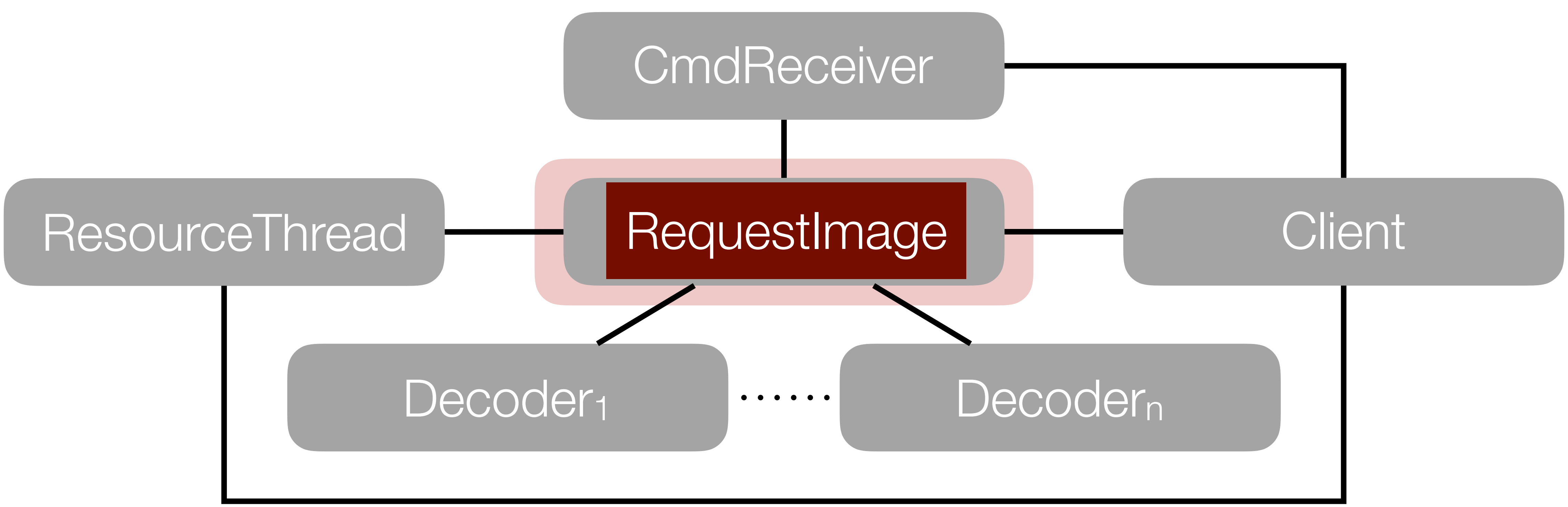
ImageCache: $\mathsf{ImgCacheCmd}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}\end{aligned}$$

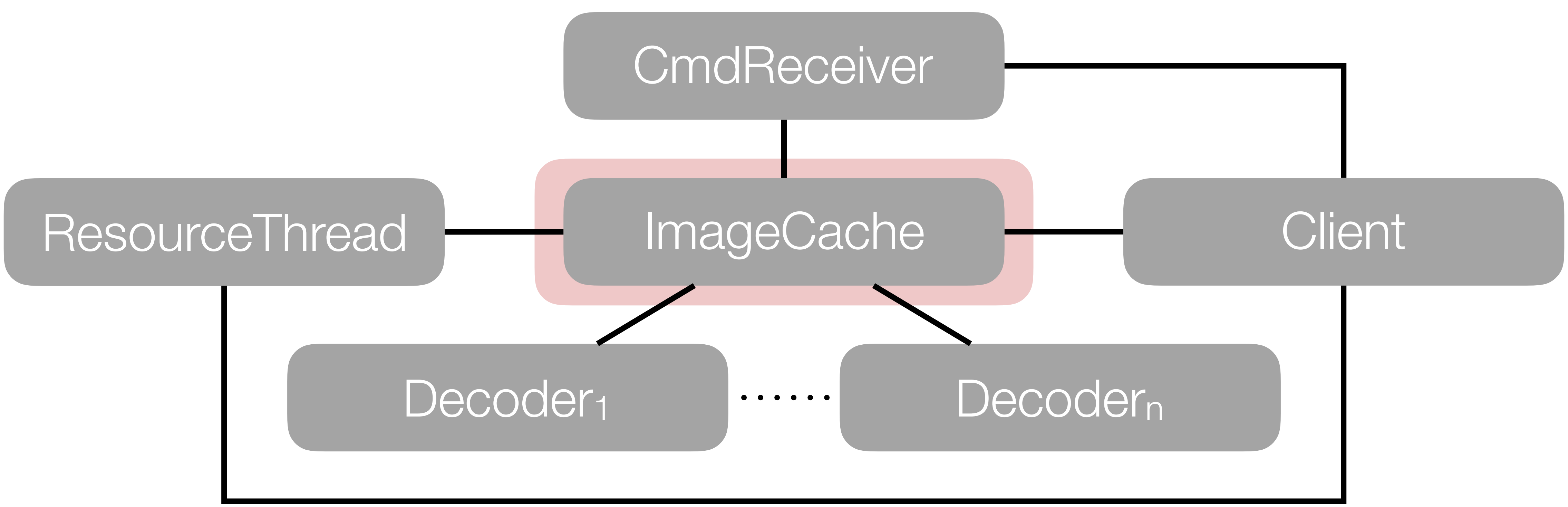
ImageCache: $\mathsf{ImgCacheCmd}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\quad\ \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}\end{aligned}$$

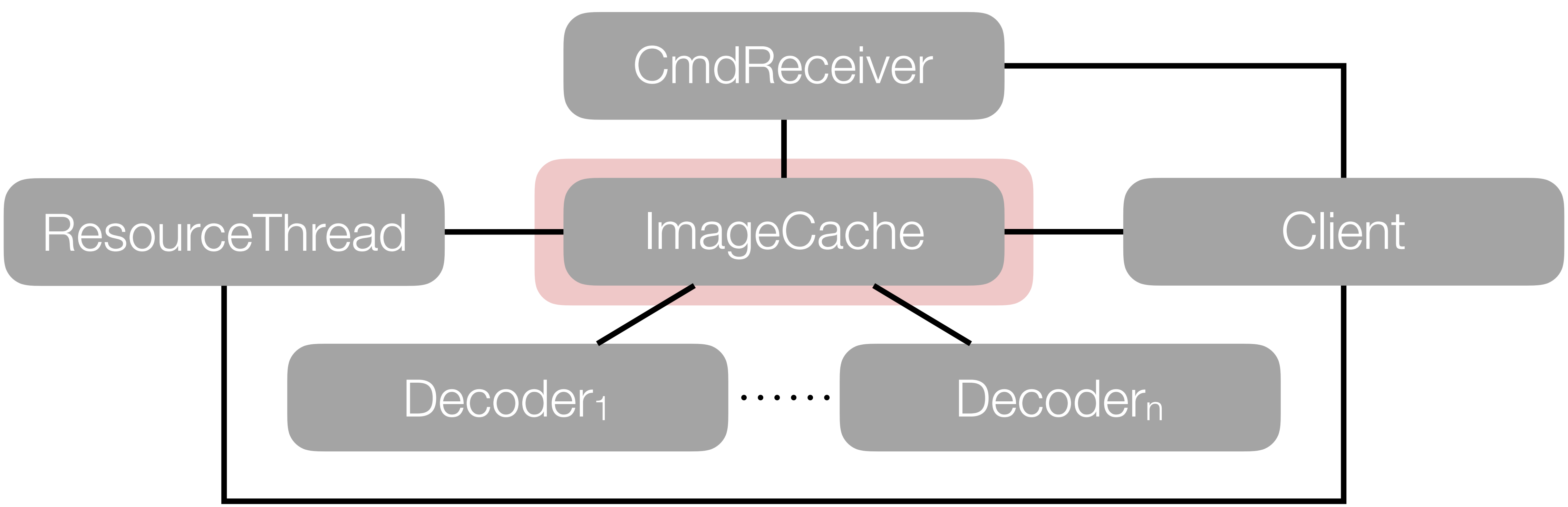
ImageCache:



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

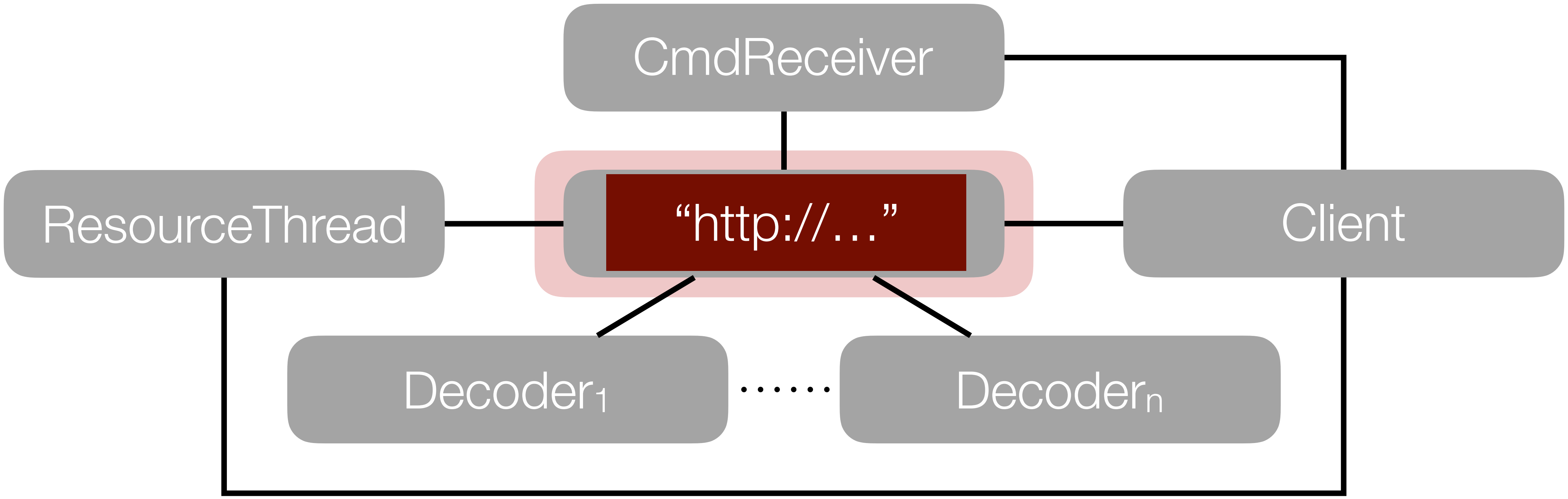
ImageCache: $\mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

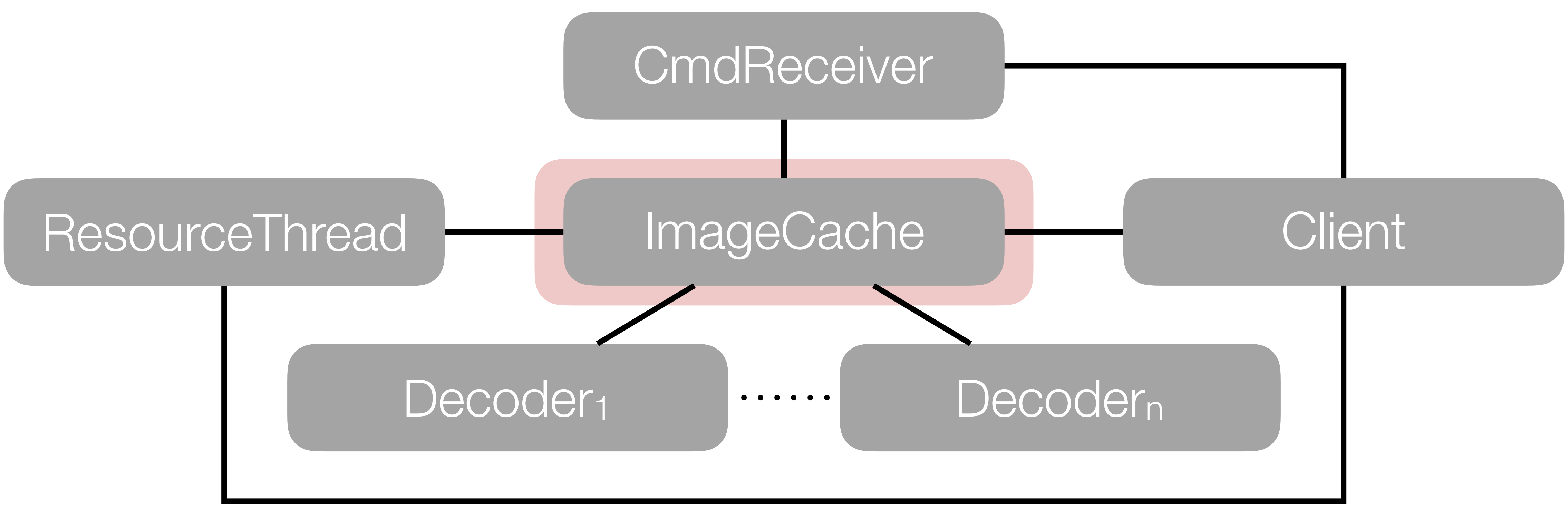
ImageCache: $\mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Session type for image loader

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\
&\ldots\\
&\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\
&\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\
\}&
\end{aligned}
$$

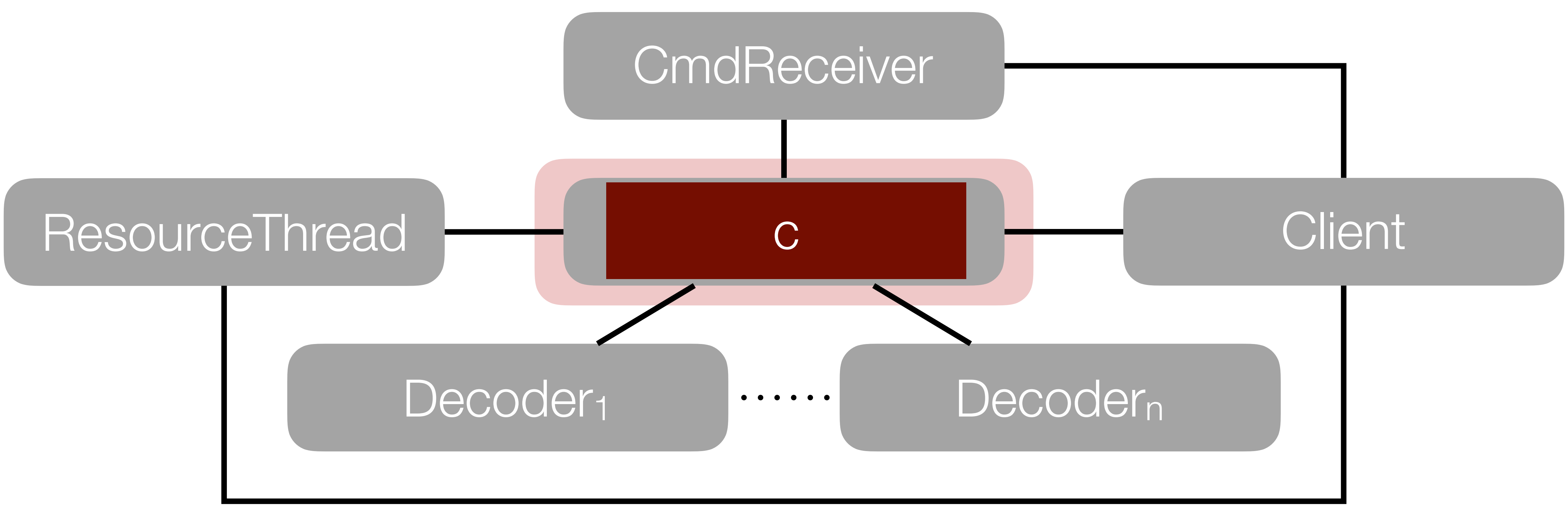
ImageCache: $\mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \ \& \ \{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus \ \{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

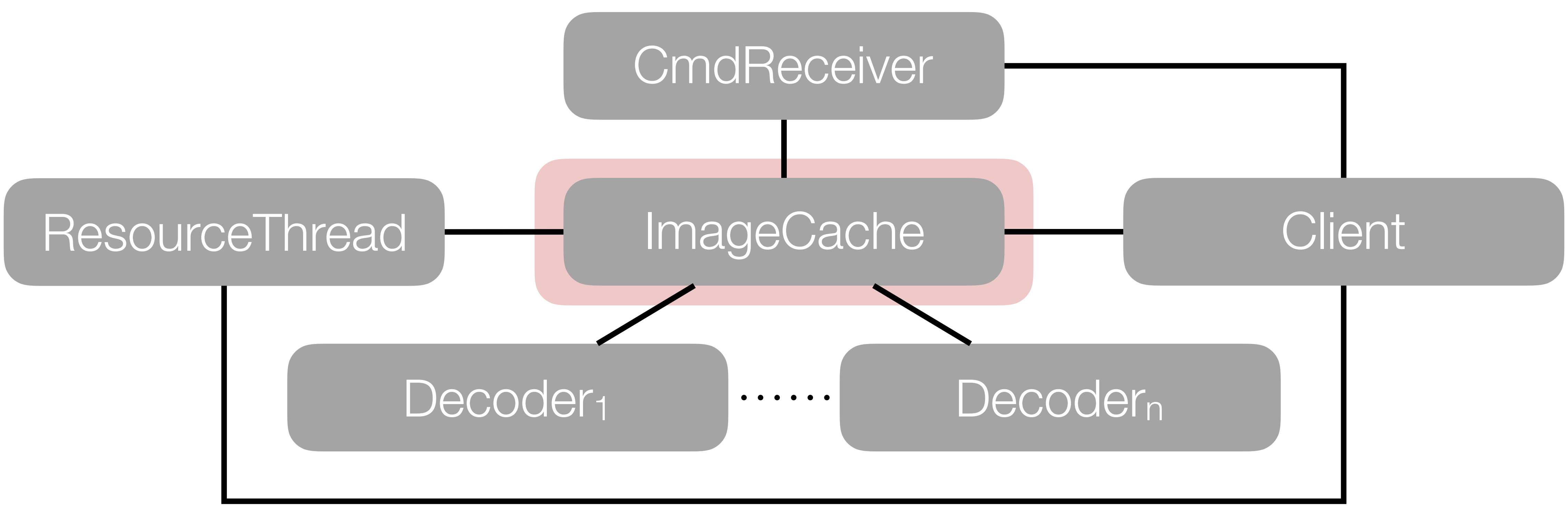
ImageCache: $\mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\ \{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\ \{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}\end{aligned}$$

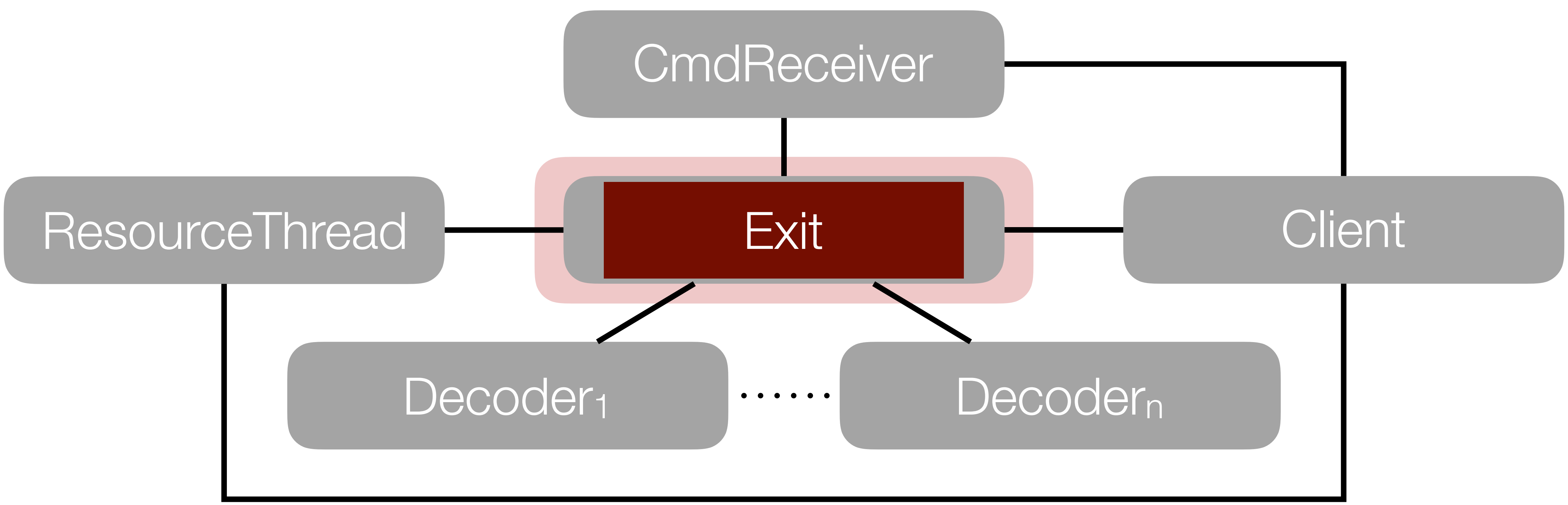
ImageCache: $\mathsf{ImgCacheCmd}$



Session type for image loader

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \& \{ & \mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\
& \ldots \\
& \mathsf{Exit} : \oplus \{ \mathsf{Running} : \mathsf{ImgCacheCmd}, \\
& \qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1} \} \\
\}
\end{aligned}
$$

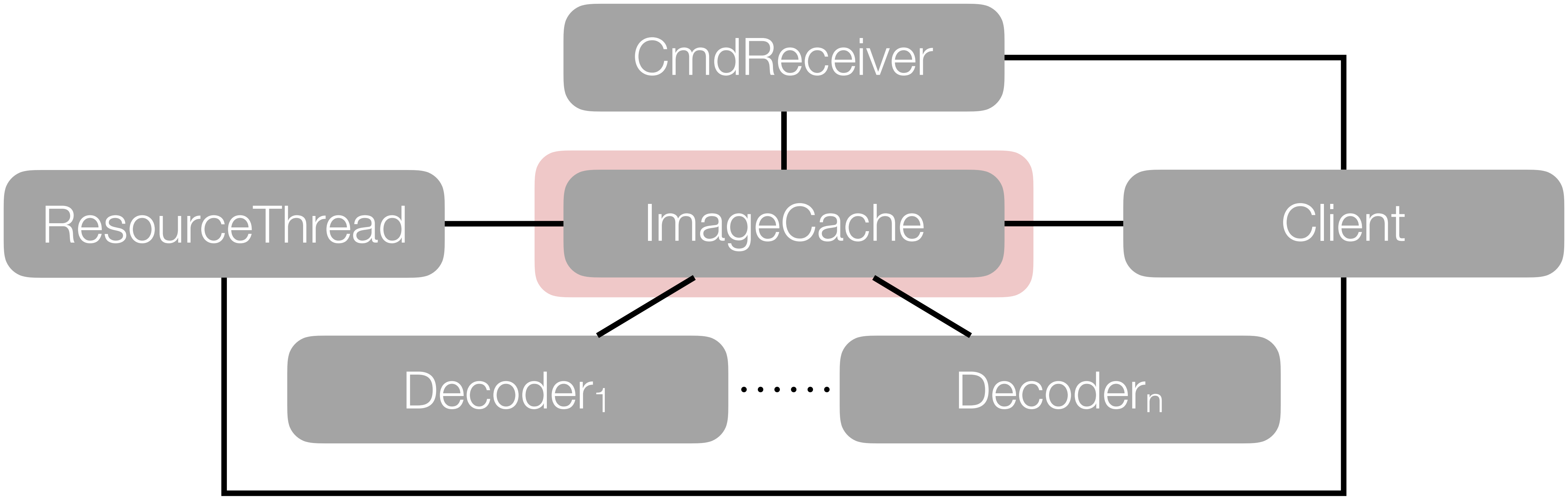
ImageCache: $\mathsf{ImgCacheCmd}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

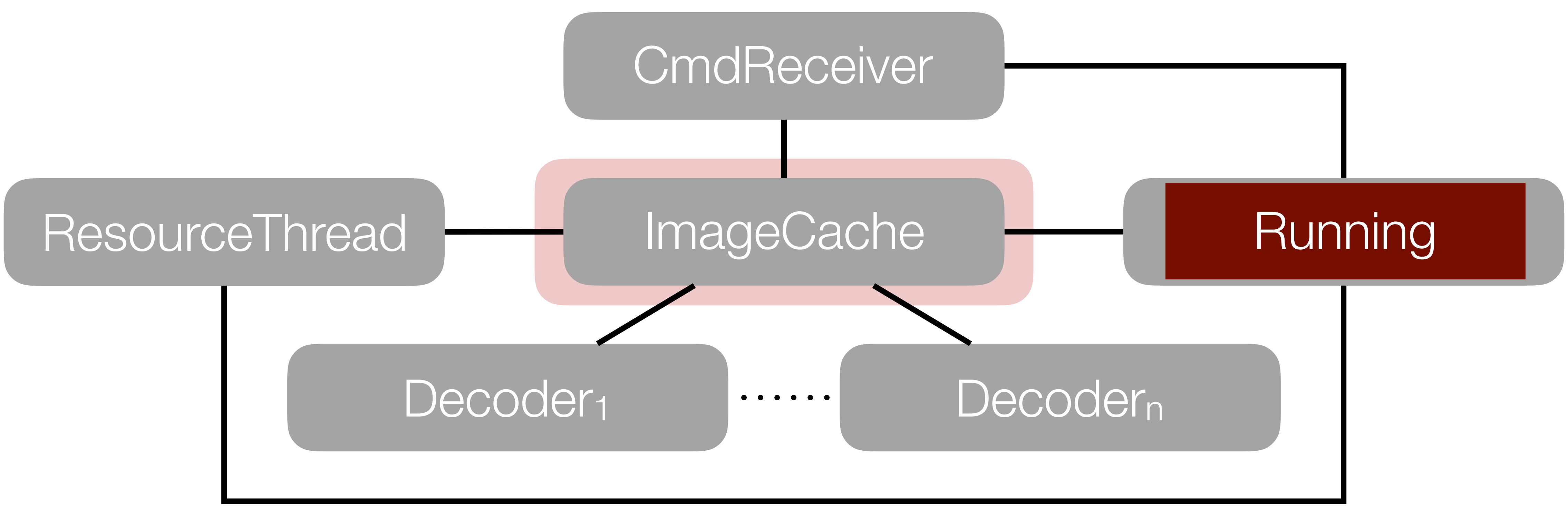
ImageCache: $\oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd}, \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}$



Session type for image loader

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}\end{aligned}$$

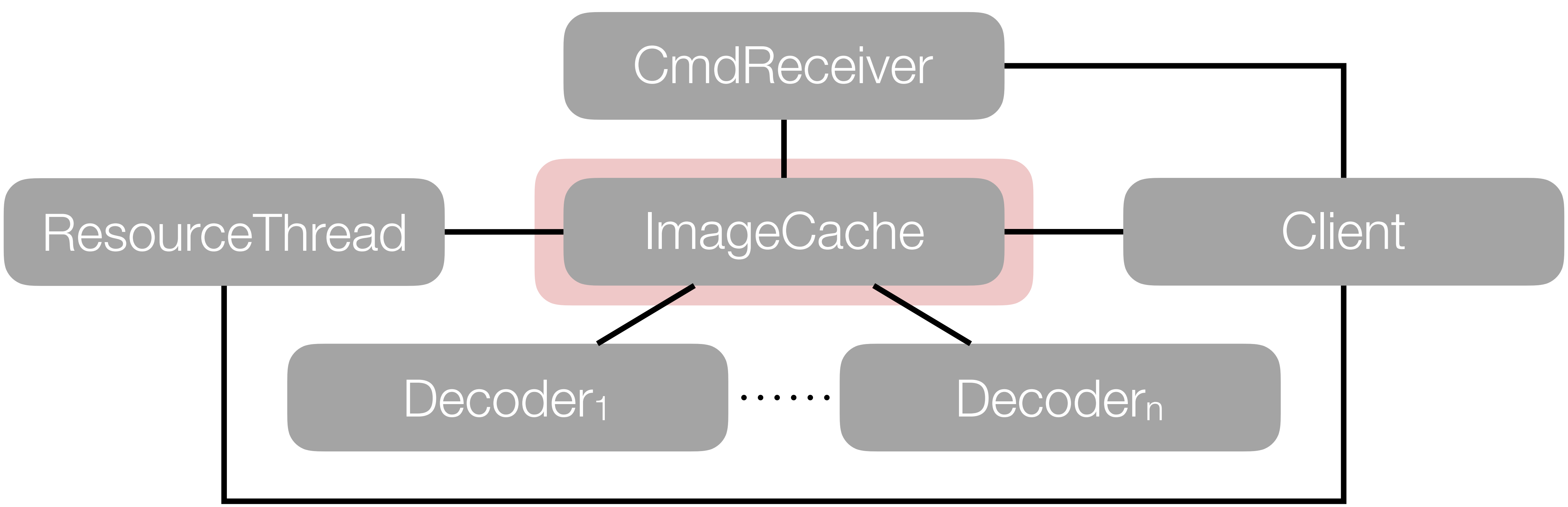
ImageCache: $\oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd}, \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}$



Session type for image loader

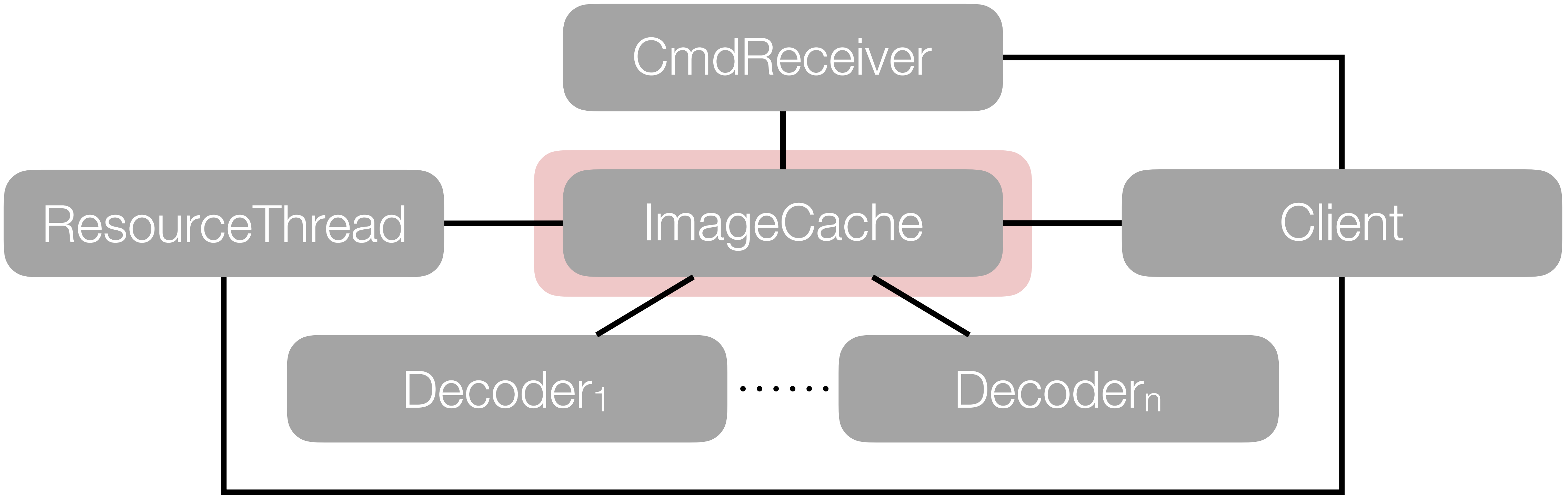
$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

ImageCache: $\mathsf{ImgCacheCmd}$



Session type for image loader

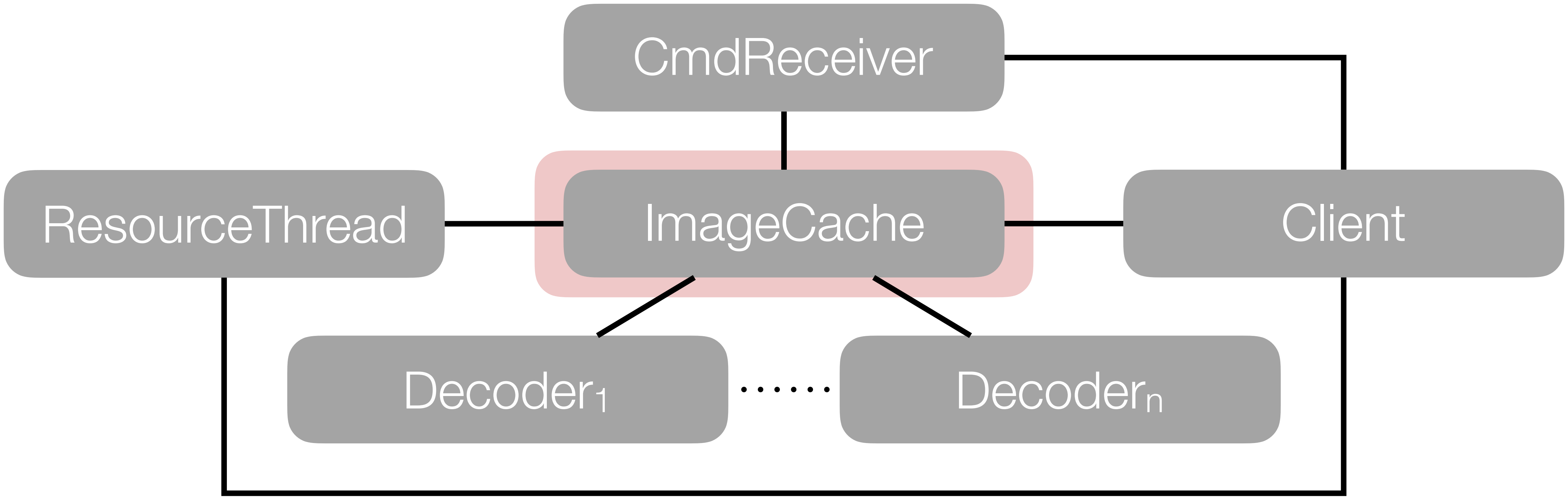
ImageCache: ImgCacheCmd



Session type for image loader

components change their session type along with message exchange

ImageCache: $\mathsf{ImgCacheCmd}$



Taking stock

Taking stock

- Session types make explicit the protocols of message exchange between concurrently executing components.

Taking stock

- Session types make explicit the protocols of message exchange between concurrently executing components.
- Typing ensures protocol adherence.

Taking stock

- Session types make explicit the protocols of message exchange between concurrently executing components.
- Typing ensures protocol adherence.
- Types make explicit interdependencies between components, enabling sequential reasoning about a component.

Taking stock

- Session types make explicit the protocols of message exchange between concurrently executing components.
- Typing ensures protocol adherence.
- Types make explicit interdependencies between components, enabling sequential reasoning about a component.

➡ **Session types are the types of message-passing concurrency.**

Session types in research and practice

Session types in research and practice

Research

- active research area since inception in 90s [Honda 1993]
- logical reconstruction based on linear logic, providing strong guarantees [Caires & Pfenning 2010, Wadler 2012]
- extension of logical session types to sharing [Balzer & Pfenning ICFP 2017, Balzer et al. CONCUR 2018, Balzer et al. ESOP 2019]

Session types in research and practice

Research

- active research area since inception in 90s [Honda 1993]
- logical reconstruction based on linear logic, providing strong guarantees [Caires & Pfenning 2010, Wadler 2012]
- extension of logical session types to sharing [Balzer & Pfenning ICFP 2017, Balzer et al. CONCUR 2018, Balzer et al. ESOP 2019]

Practice

- Lightweight integration of session types or session libraries (with varying static guarantees) into Scala, Java, Haskell, OCaml, Go, Rust, Python.
- Collaboration with Mozilla Research on integrating our work.

Session types in research and practice

Research

- active research area since inception in 90s [Honda 1993]
- logical reconstruction based on linear logic, providing strong guarantees [Caires & Pfenning 2010, Wadler 2012]
- extension of logical session types to sharing [Balzer & Pfenning ICFP 2017, Balzer et al. CONCUR 2018, Balzer et al. ESOP 2019]

Practice

- Lightweight integration of session types or session libraries (with varying static guarantees) into Scala, Java, Haskell, OCaml, Go, Rust, Python.
- Collaboration with Mozilla Research on integrating our work.

Logic-based shared session types

Linear logic session types

Linear logic session types

Provide strong guarantees:

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

Linear logic session types

exactly one client

Provide strong guarantees:

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

Linear logic session types

Provide strong guarantees:

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

exactly one client

processes graph forms
a tree at run-time

Linear logic session types

Provide strong guarantees

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

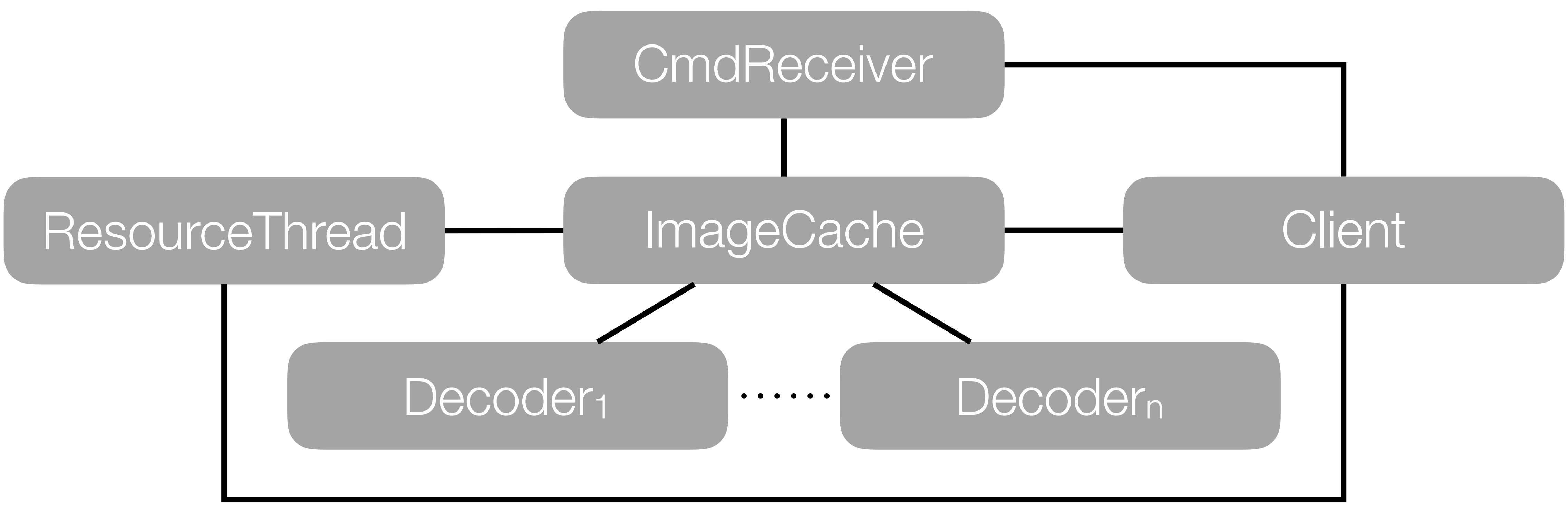
But, they rule out sharing

Linear logic session types

Provide strong guarantees

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

But, they rule out sharing

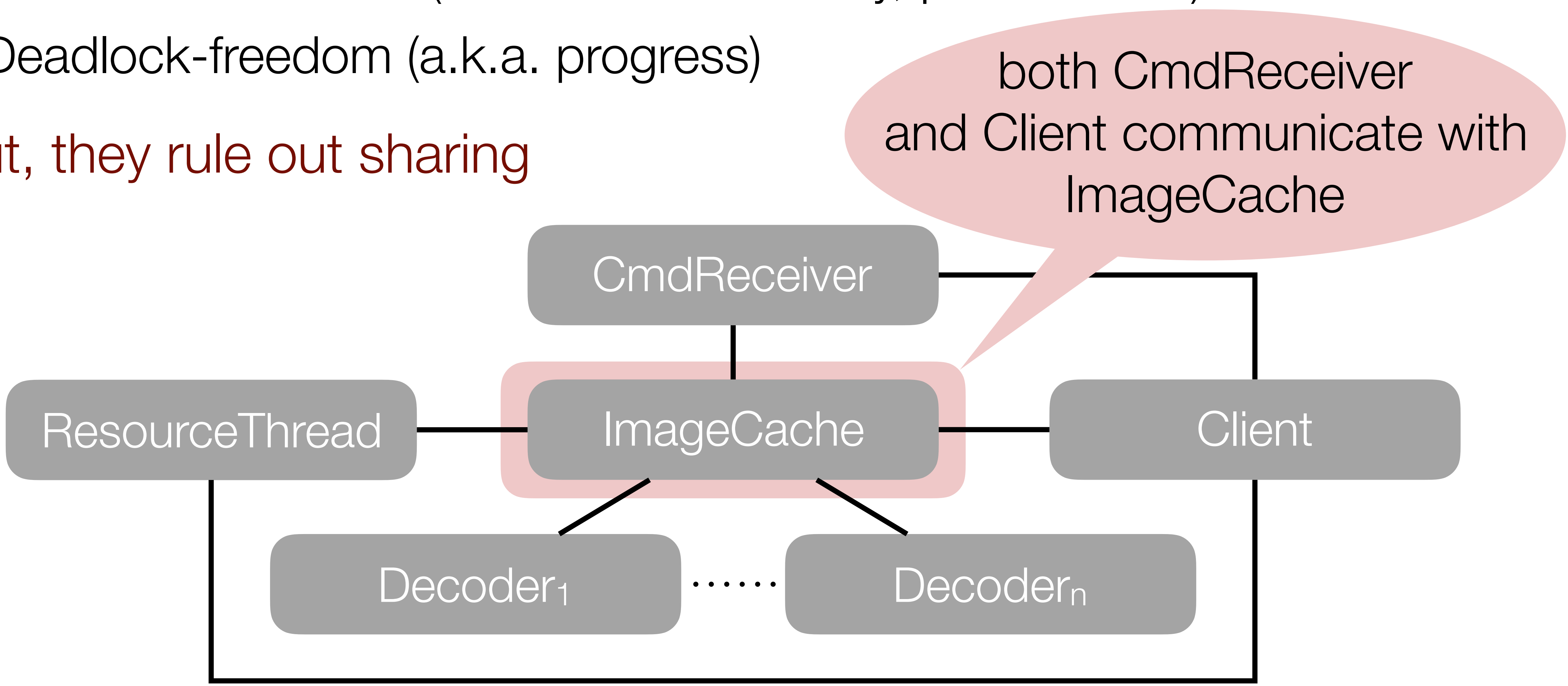


Linear logic session types

Provide strong guarantees

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

But, they rule out sharing



Linear logic session types

Provide strong guarantees

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

But, they rule out sharing

→ Linear logic session types cannot accommodate certain practical programming scenarios.

Linear logic session types

Provide strong guarantees

- Data-race-freedom
- Protocol adherence (a.k.a. session fidelity, preservation)
- Deadlock-freedom (a.k.a. progress)

But, they rule out sharing

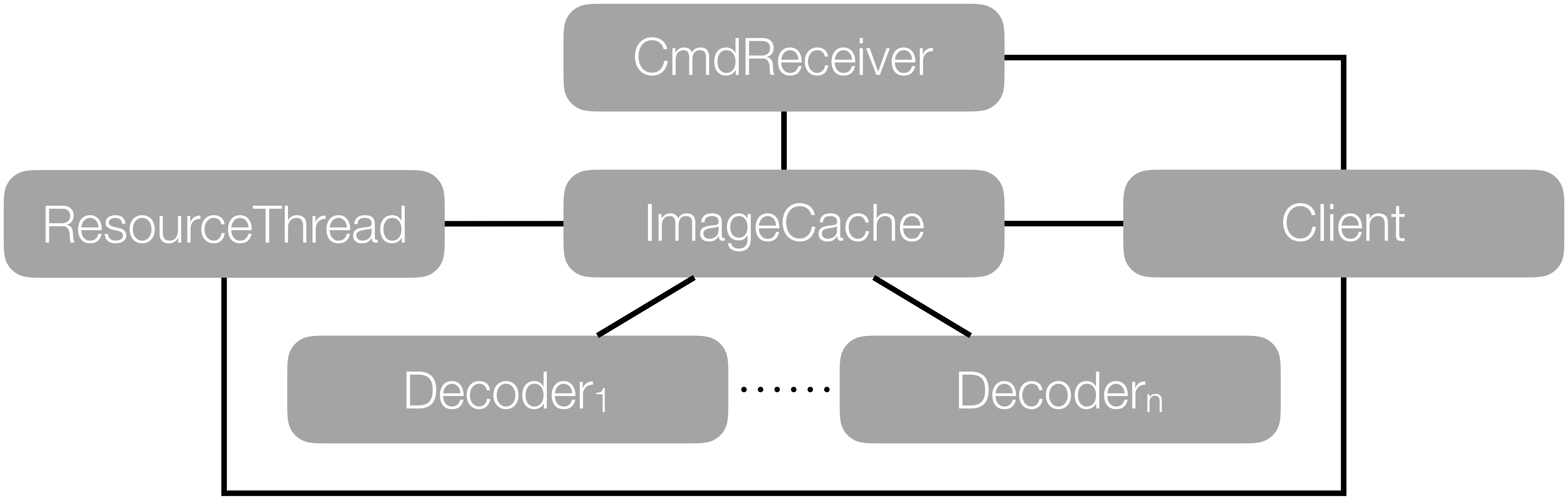
➡ Linear logic session types cannot accommodate certain practical programming scenarios.

➡ Let's introduce sharing while maintaining above guarantees.

Challenges of sharing

Challenges of sharing

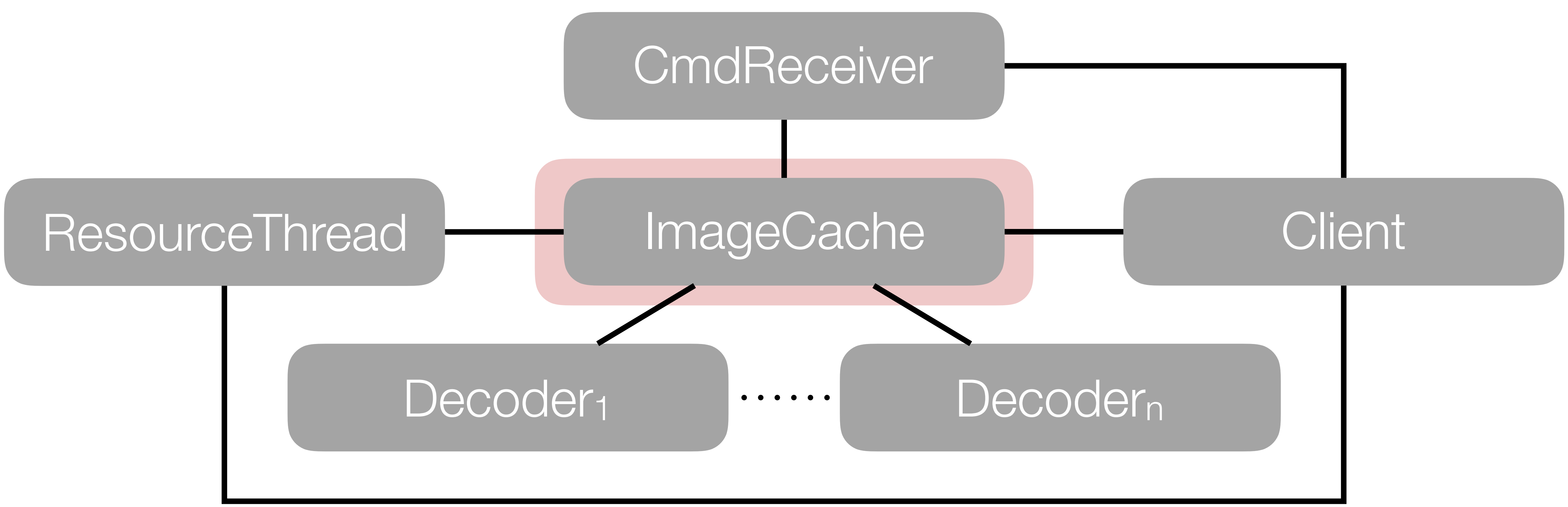
$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \mathbin{\&} \{ & \mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\
& \ldots \\
& \mathsf{Exit} : \oplus \{ \mathsf{Running} : \mathsf{ImgCacheCmd}, \\
& \qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1} \} \\
\} &
\end{aligned}
$$



Challenges of sharing

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\, \{ & \mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\
& \ldots \\
& \mathsf{Exit} : \oplus \{ \mathsf{Running} : \mathsf{ImgCacheCmd}, \\
& \qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1} \} \\
\}
\end{aligned}
$$

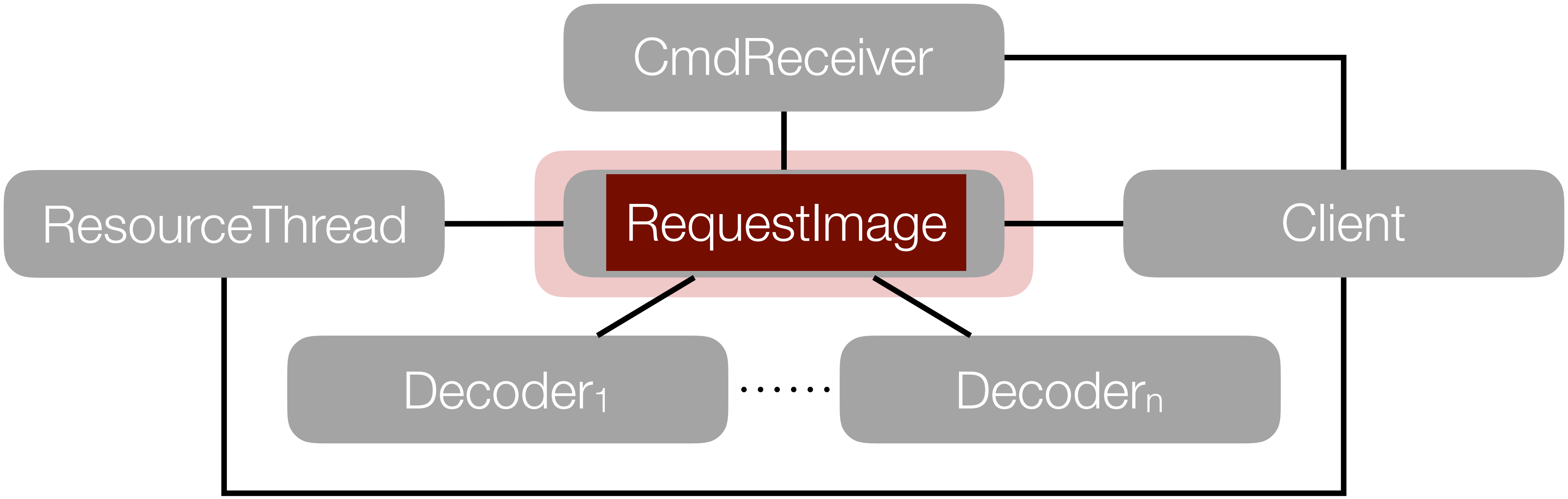
ImageCache: $\mathsf{ImgCacheCmd}$



Challenges of sharing

$$\begin{aligned}\mathsf{ImgCacheCmd} = \& \{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus \{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

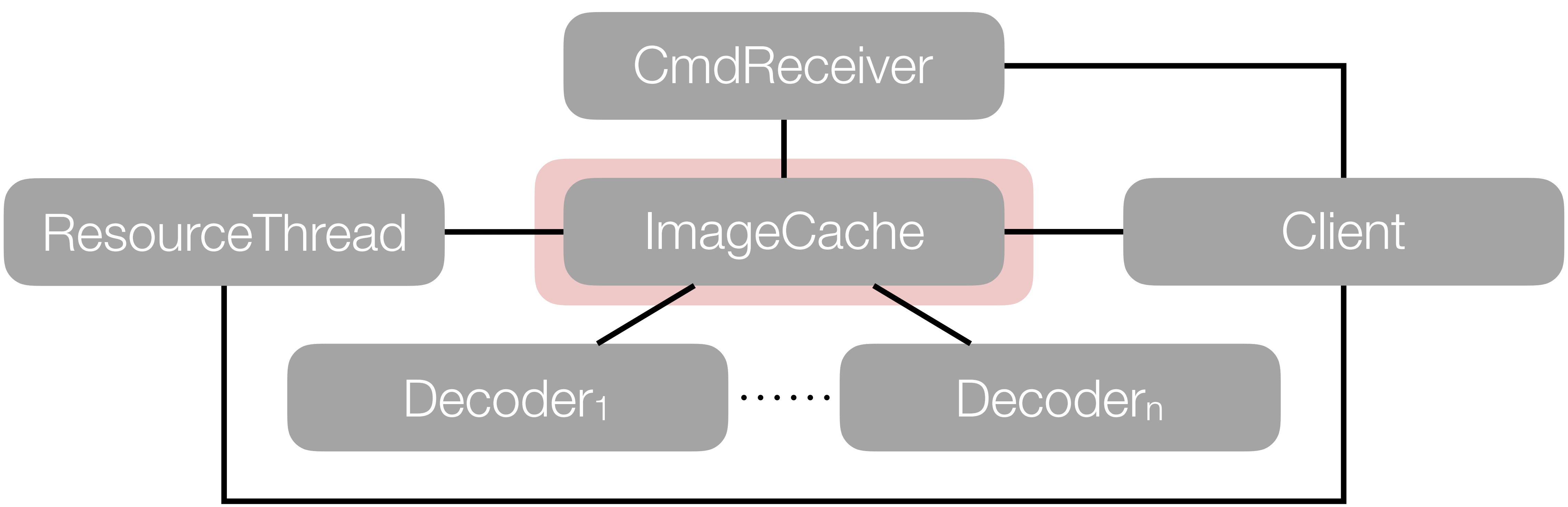
ImageCache: $\mathsf{ImgCacheCmd}$



Challenges of sharing

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\
&\ldots\\
&\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\
&\qquad\qquad\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\
\}&
\end{aligned}
$$

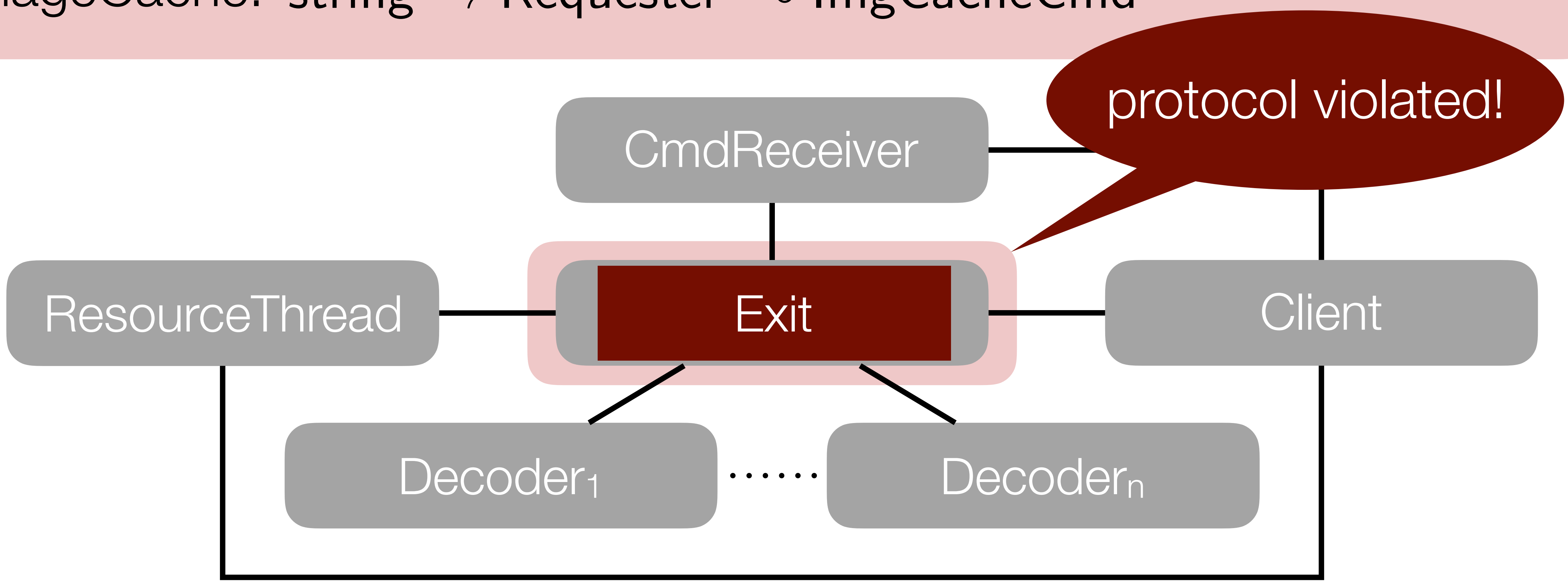
ImageCache: $\mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Challenges of sharing

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

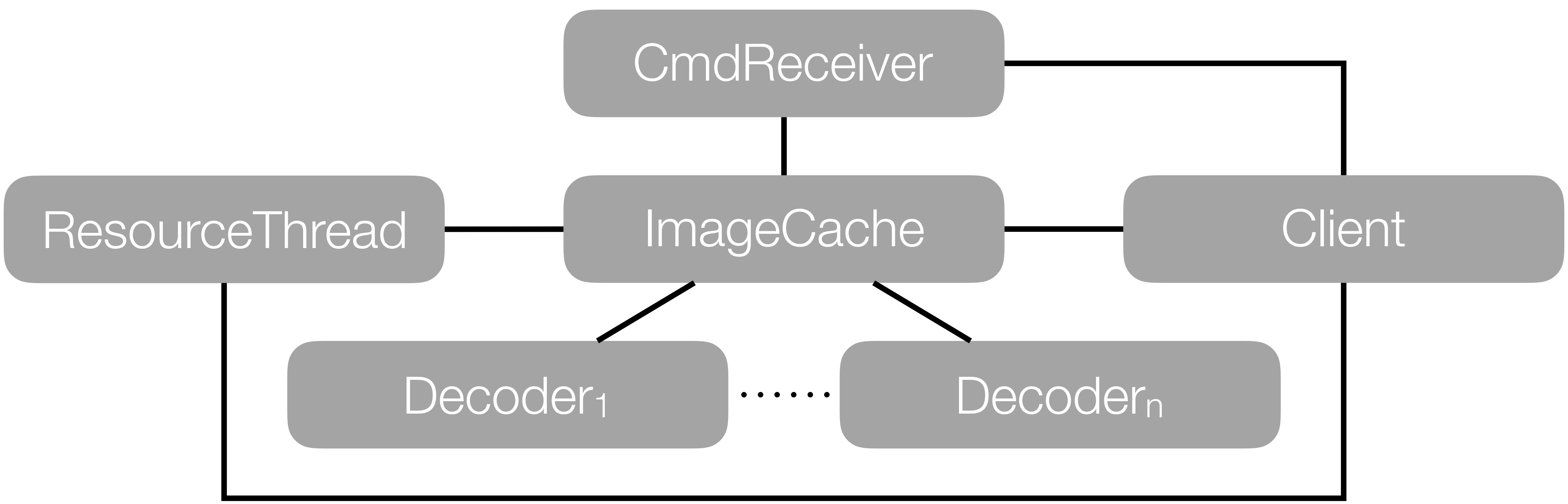
ImageCache: $\mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Challenges of sharing

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\,\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\ldots\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\\ \}&\end{aligned}$$

How to restore protocol adherence in the presence of sharing (a.k.a. aliasing)?



Idea: acquire-release

Idea: acquire-release

Clients of shared channels must communicate along that channel in mutual exclusion from each other.

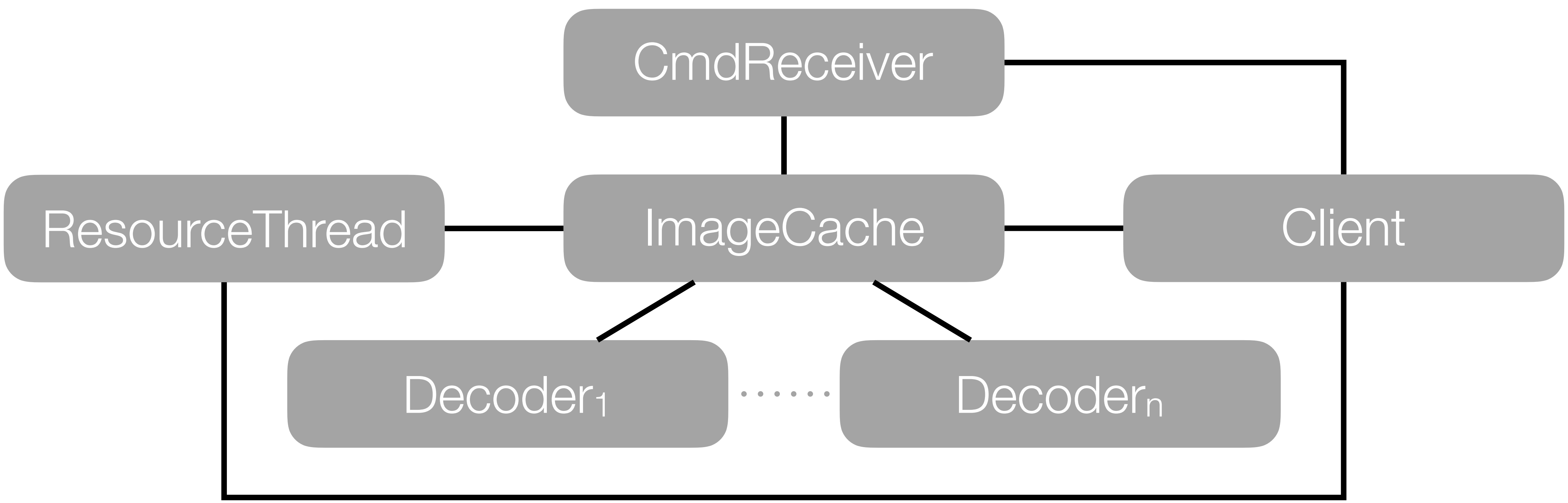
Idea: acquire-release

Clients of shared channels must communicate along that channel in mutual exclusion from each other.

Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.

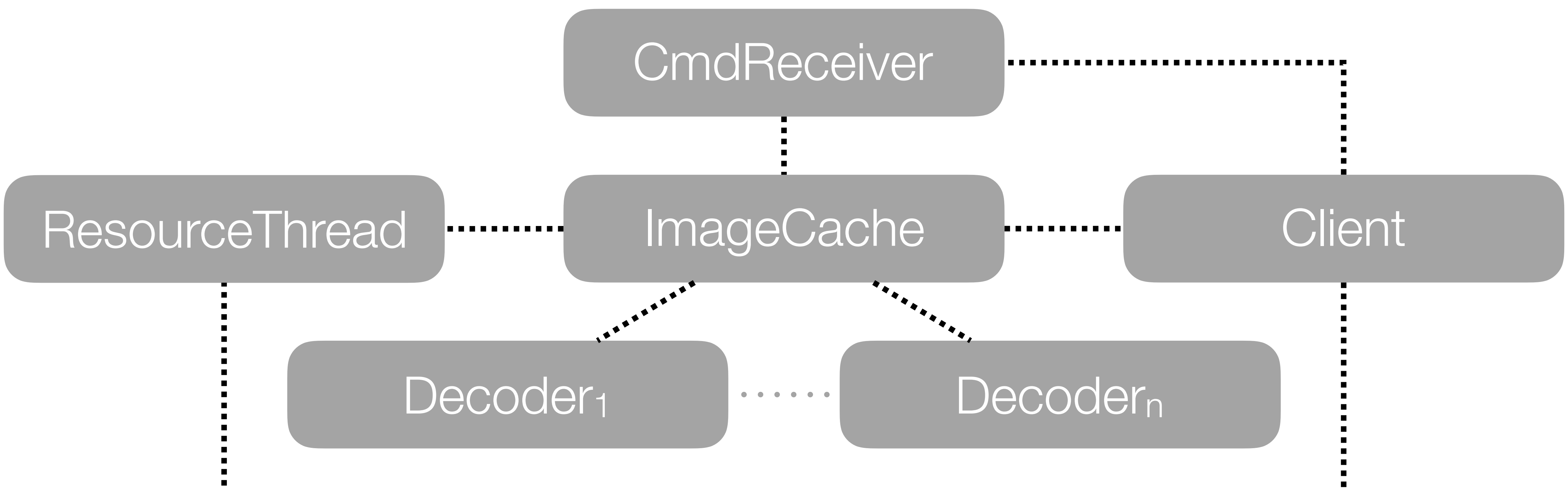
Idea: acquire-release

- Clients of shared channels must communicate along that channel in mutual exclusion from each other.
- Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.



Idea: acquire-release

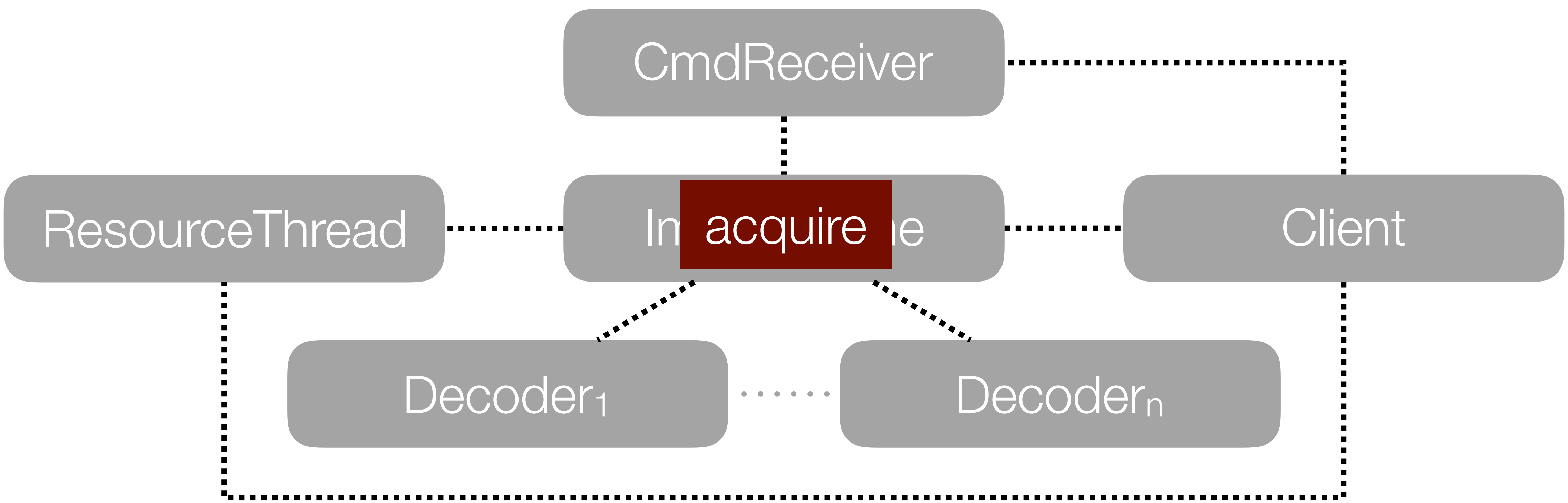
- Clients of shared channels must communicate along that channel in mutual exclusion from each other.
- Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.



Legend: ······· shared channel

Idea: acquire-release

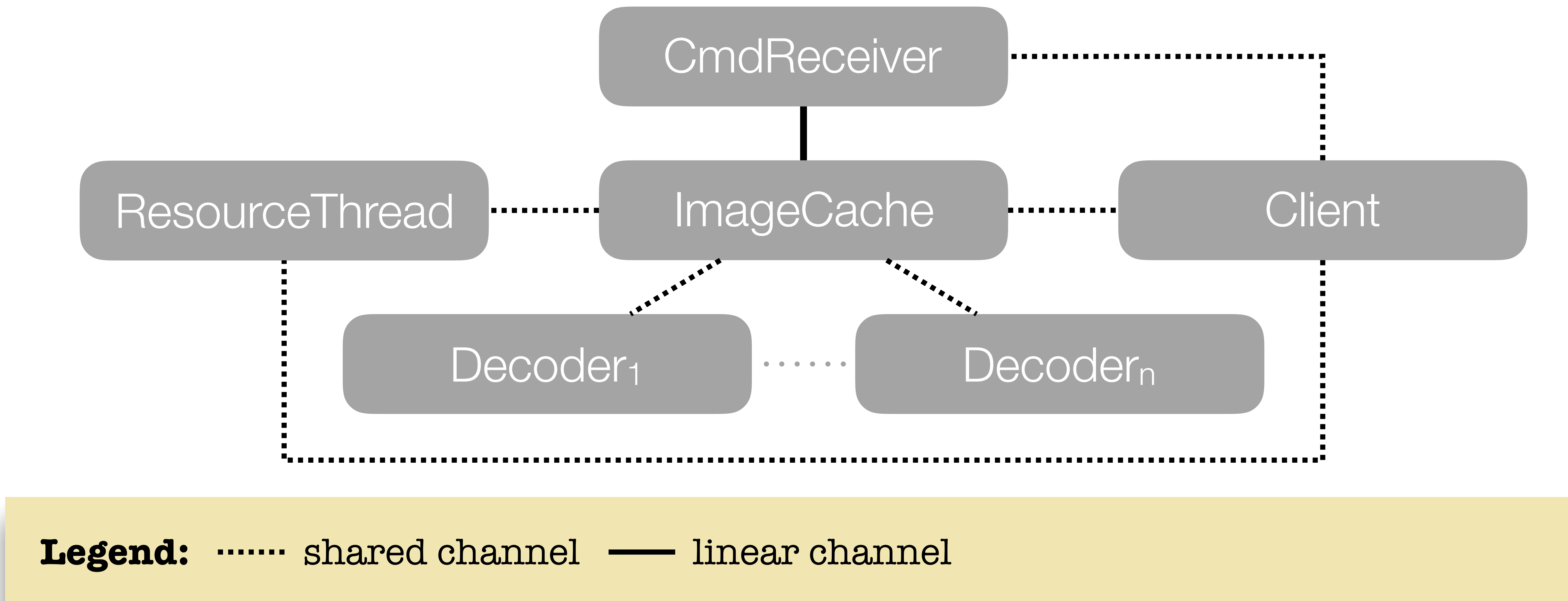
- Clients of shared channels must communicate along that channel in mutual exclusion from each other.
- Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.



Legend: ······· shared channel

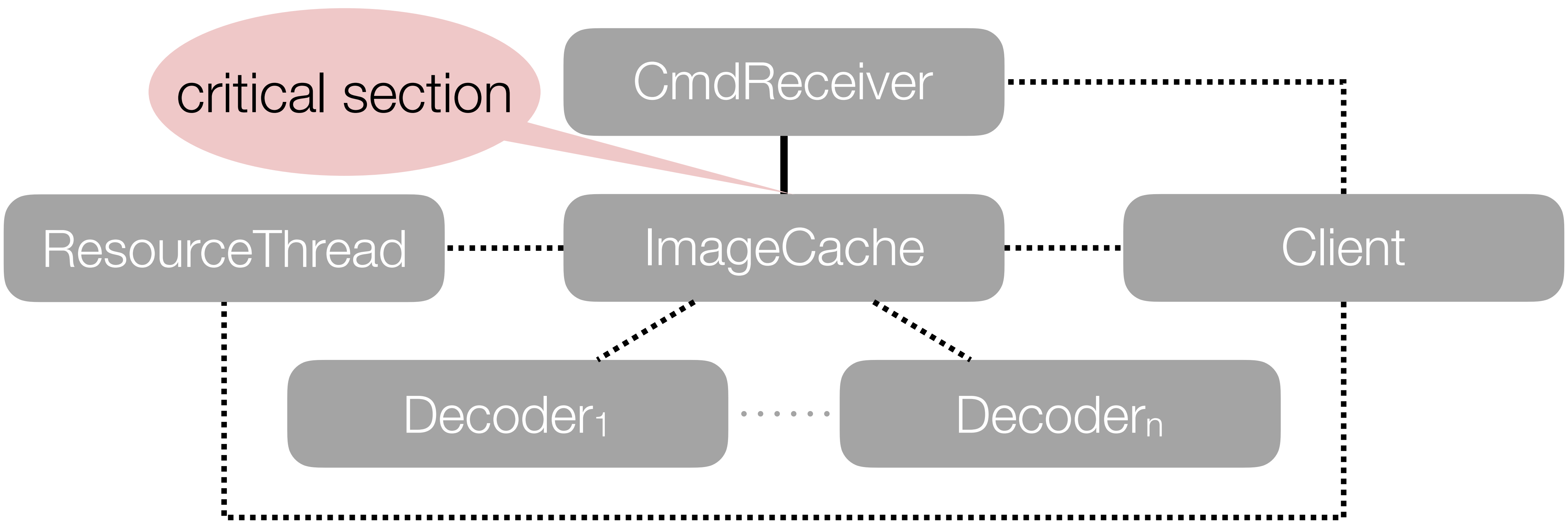
Idea: acquire-release

- Clients of shared channels must communicate along that channel in mutual exclusion from each other.
- Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.



Idea: acquire-release

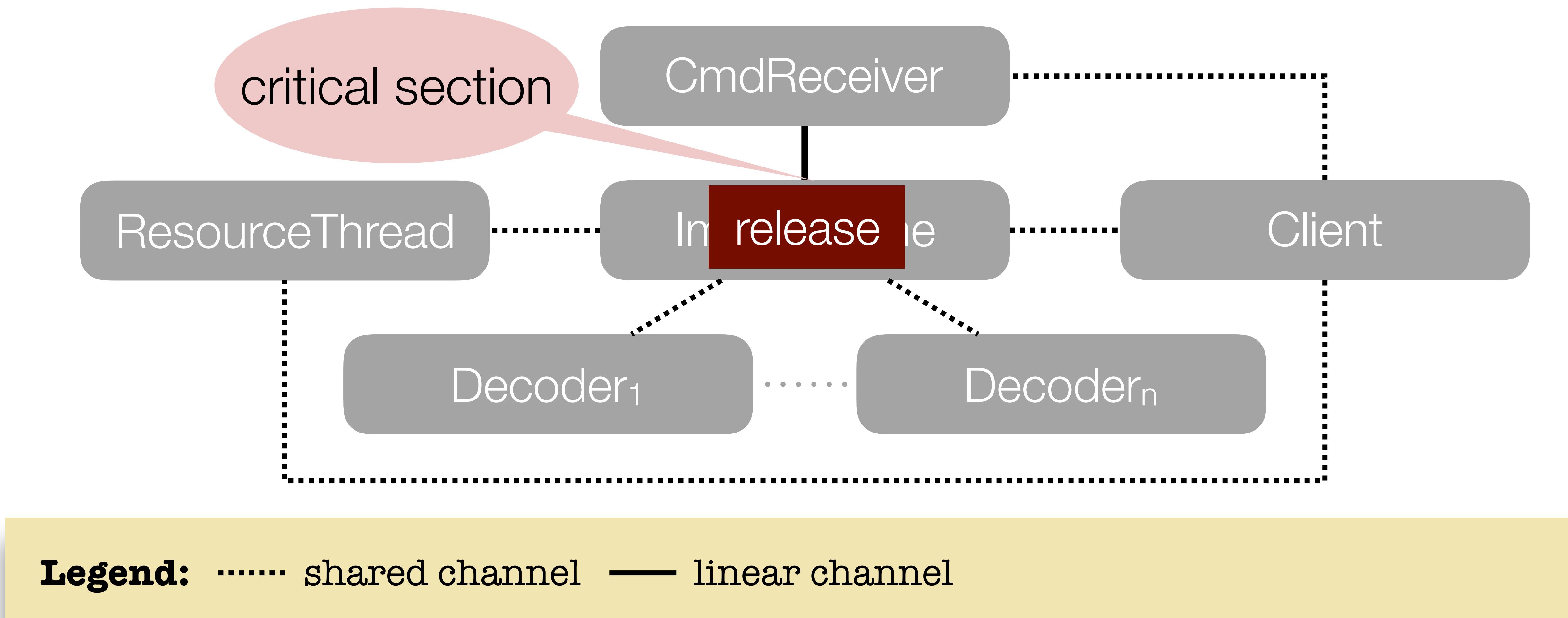
- Clients of shared channels must communicate along that channel in mutual exclusion from each other.
- Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.



Legend: ······· shared channel —— linear channel

Idea: acquire-release

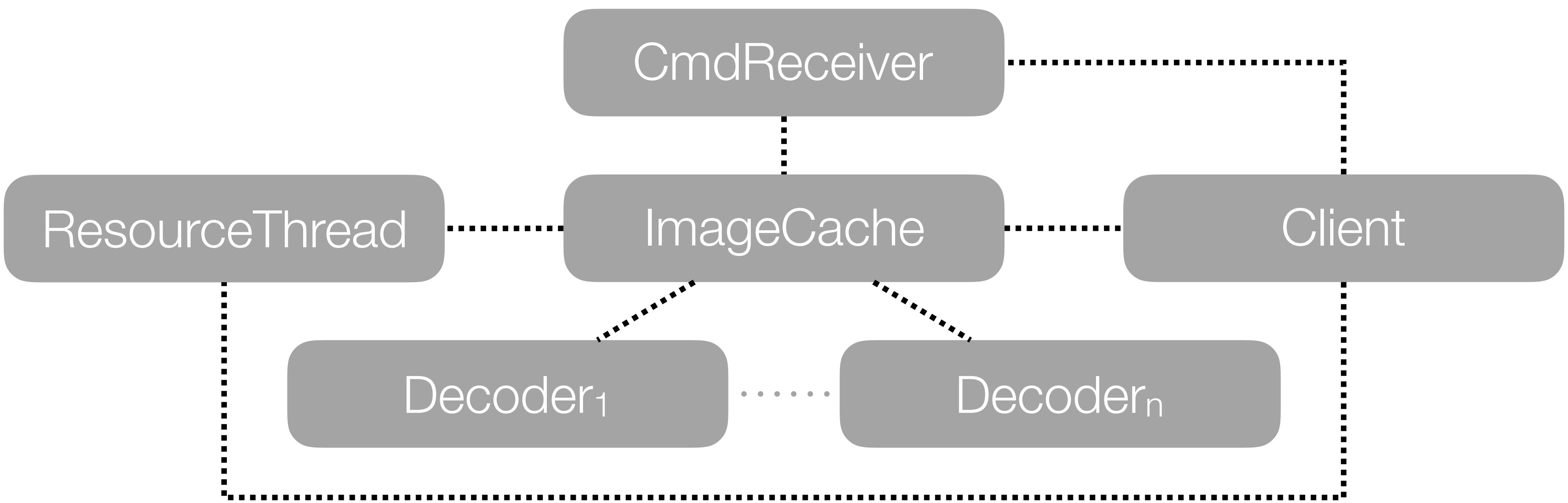
- Clients of shared channels must communicate along that channel in mutual exclusion from each other.
- Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.



Idea: acquire-release

Clients of shared channels must communicate along that channel in mutual exclusion from each other.

Acquiring a shared channel gives exclusive access, releasing an acquired channel relinquishes exclusive access.

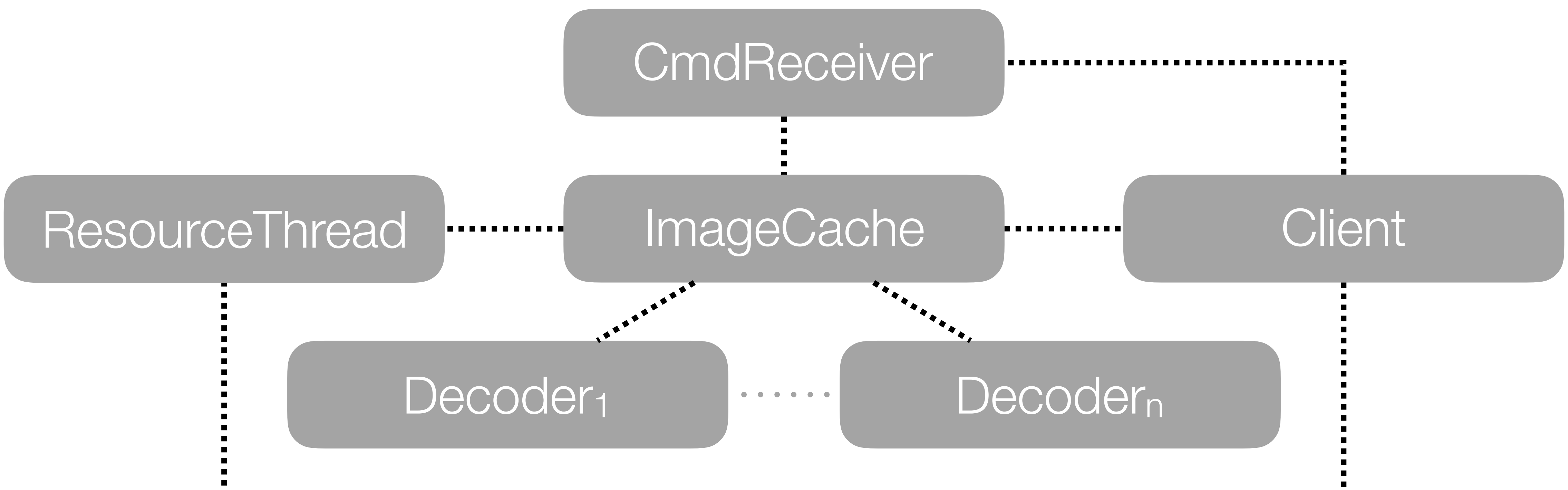


Legend: ······· shared channel ——— linear channel

Have we restored protocol adherence?

Have we restored protocol adherence?

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}\end{aligned}$$

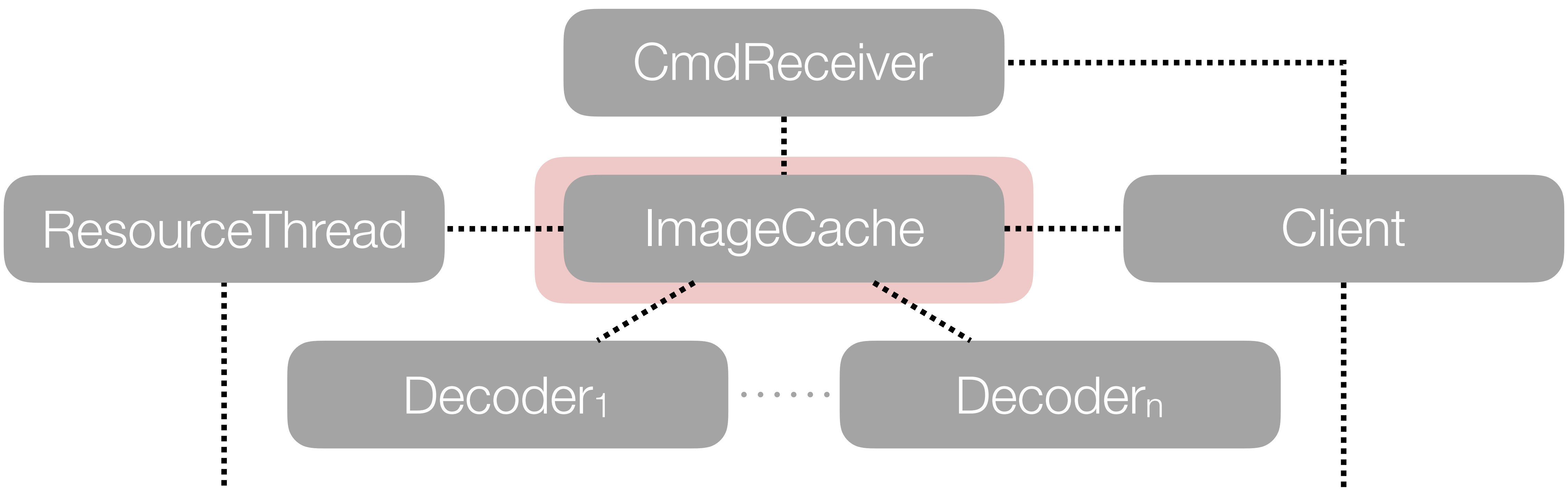


Legend: ······· shared channel ——— linear channel

Have we restored protocol adherence?

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\, \{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}\end{aligned}$$

ImageCache: $\mathsf{ImgCacheCmd}$

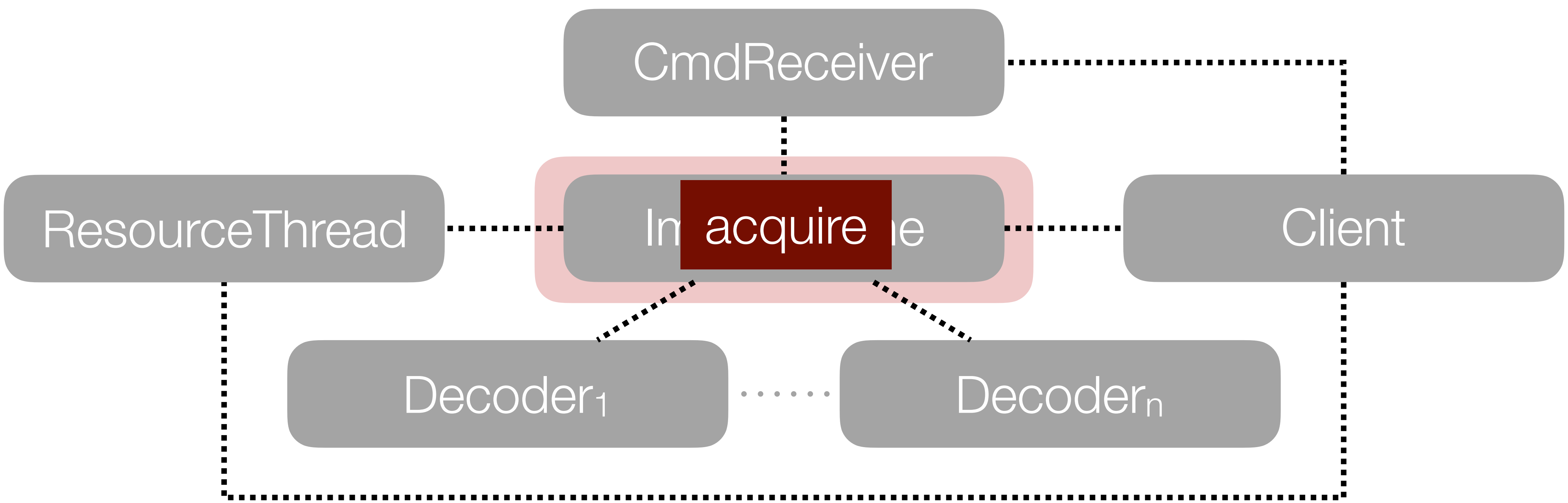


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\
&\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\
&\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}
\end{aligned}
$$

ImageCache: $\mathsf{ImgCacheCmd}$

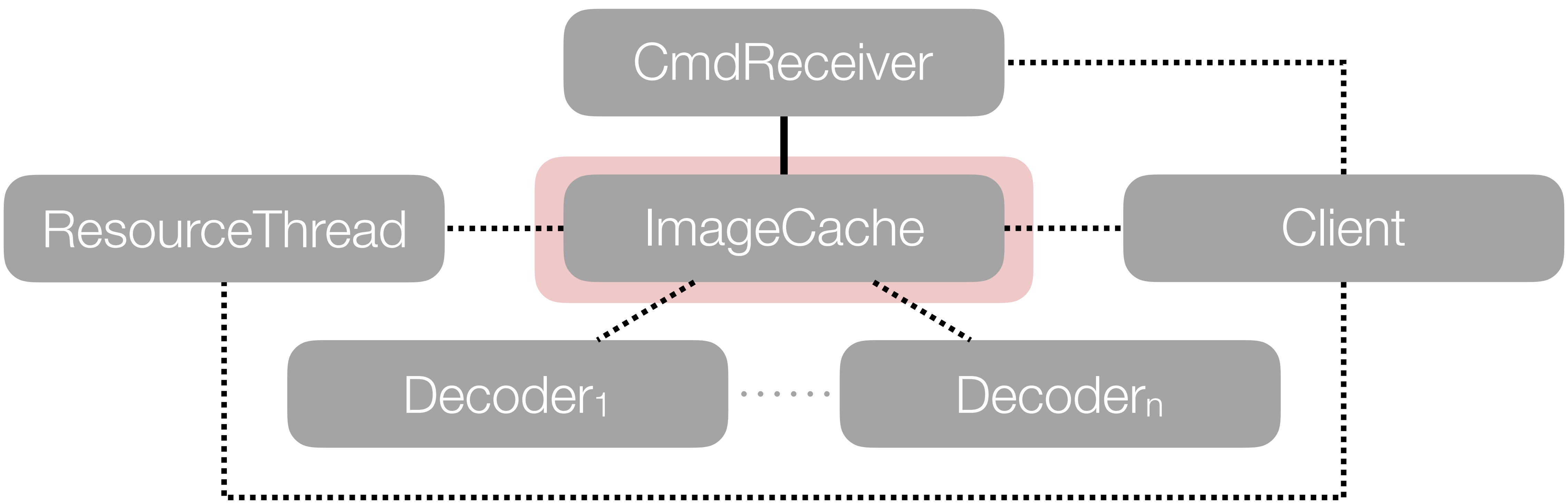


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\mathsf{ImgCacheCmd} = \&\{\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\ \mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd}, \\ \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}$$

ImageCache: $\mathsf{ImgCacheCmd}$

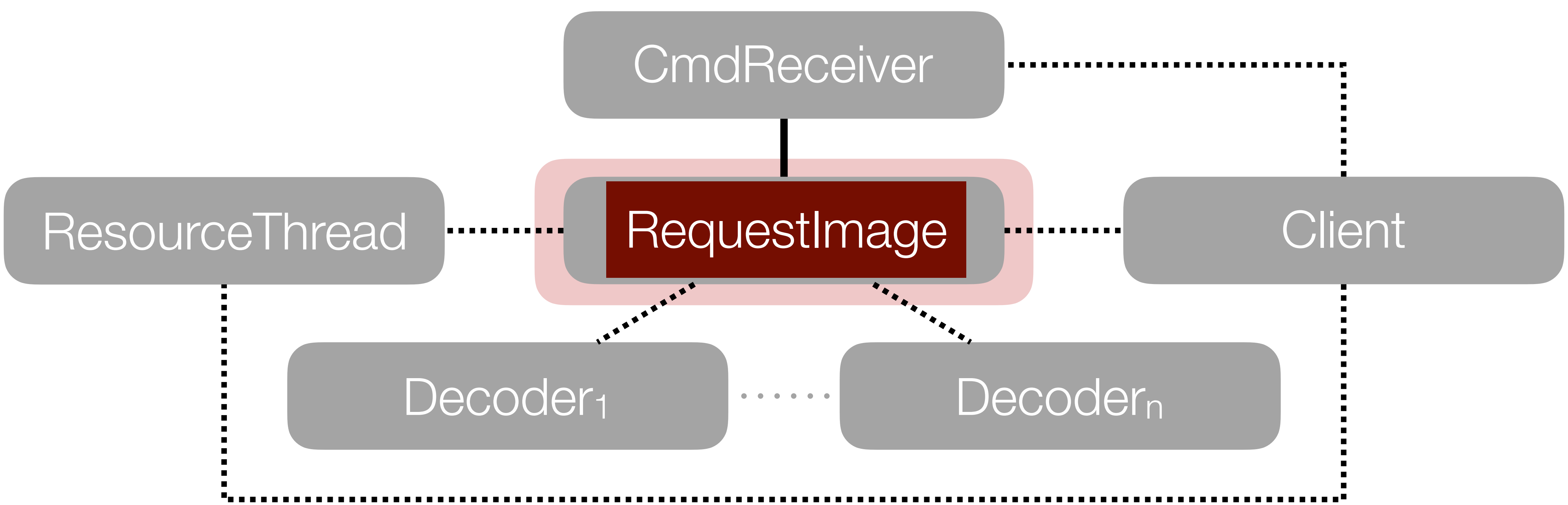


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\mathsf{ImgCacheCmd} = \&\{\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ \mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}$$

ImageCache: $\mathsf{ImgCacheCmd}$

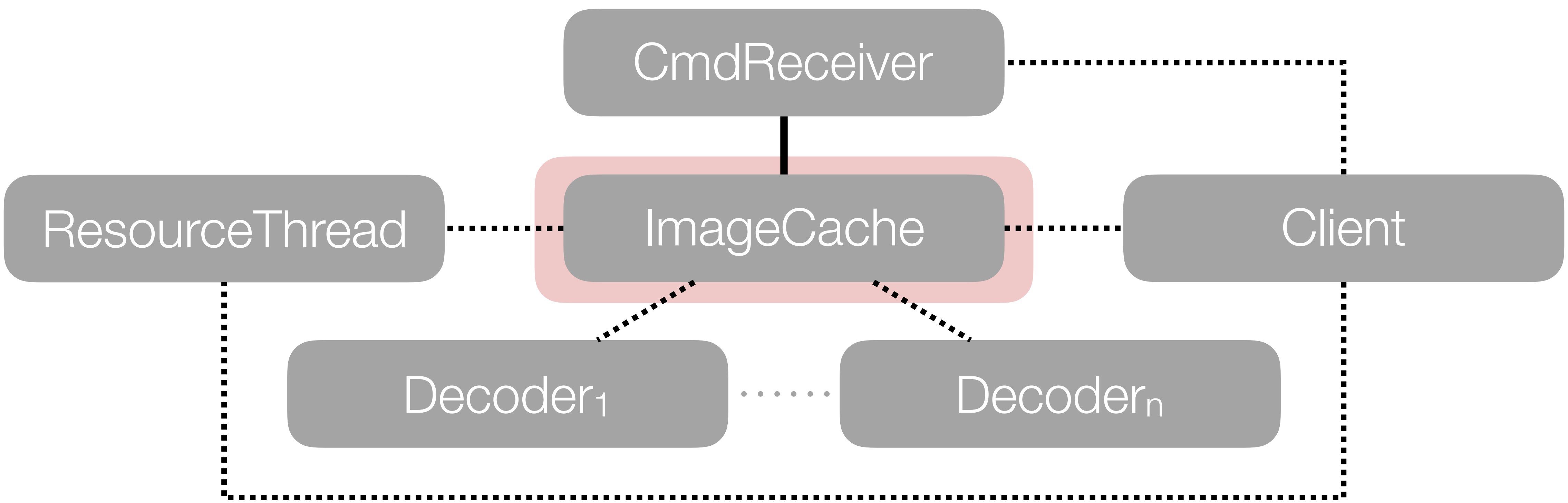


Legend: ······ shared channel —— linear channel

Have we restored protocol adherence?

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}\end{aligned}$$

ImageCache: $\mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$

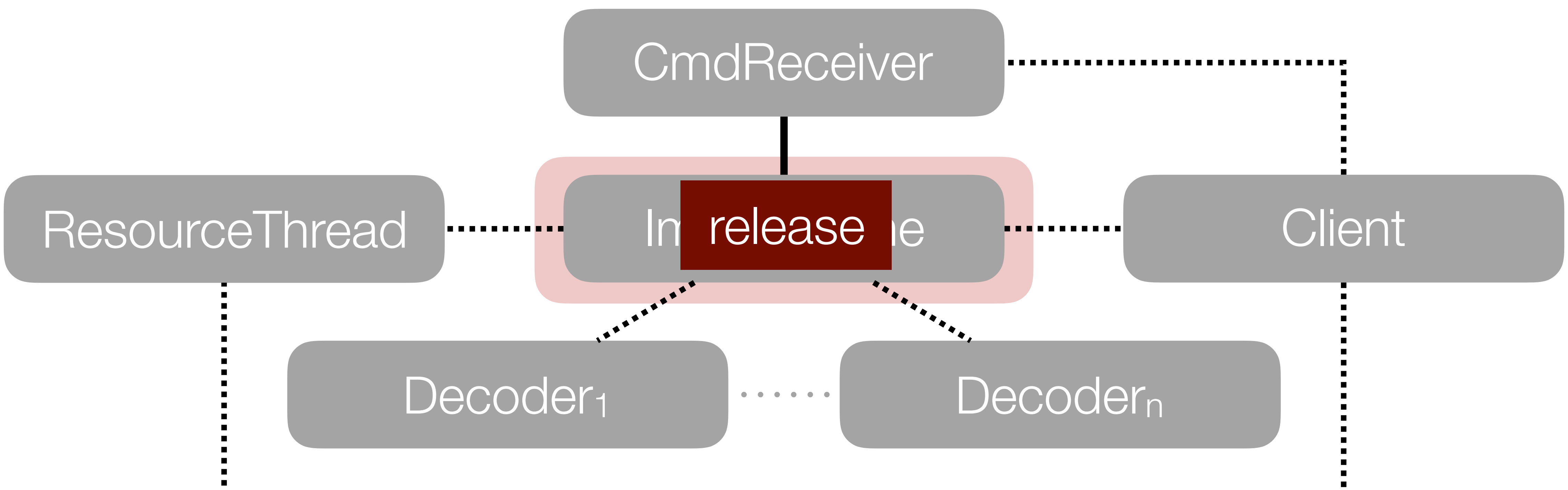


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\mathsf{ImgCacheCmd} = \&\{\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},$$
$$\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},$$
$$\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}$$

ImageCache: $\mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$

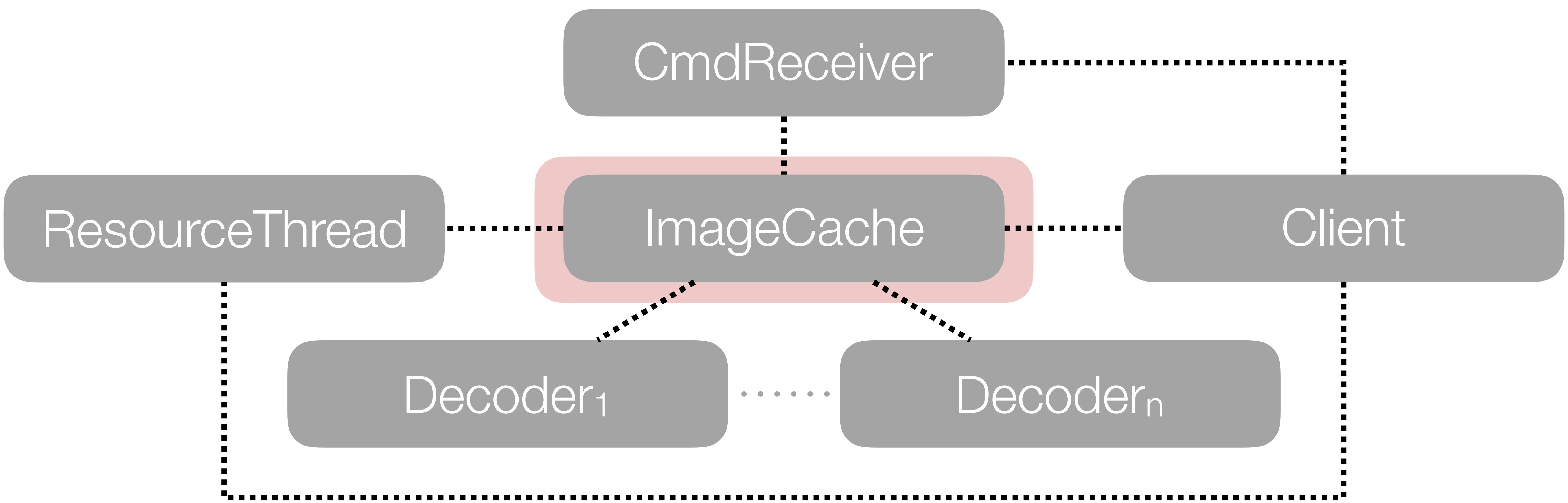


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\qquad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}\end{aligned}$$

ImageCache: $\mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$

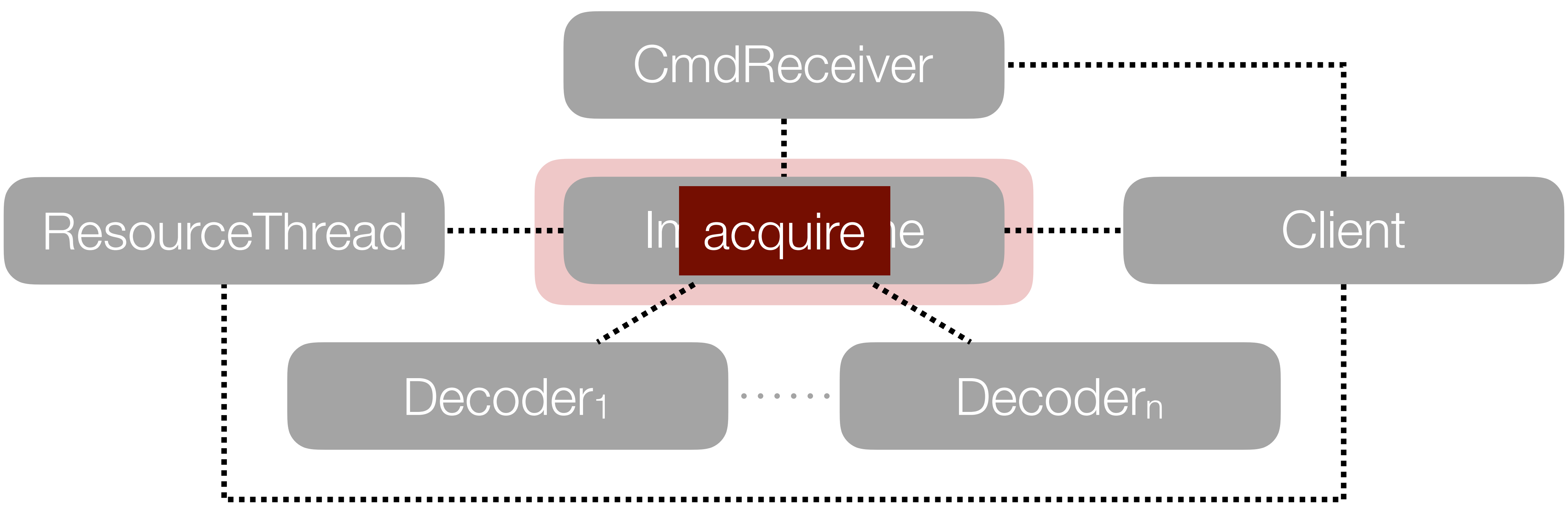


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}\end{aligned}$$

ImageCache: $\mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$

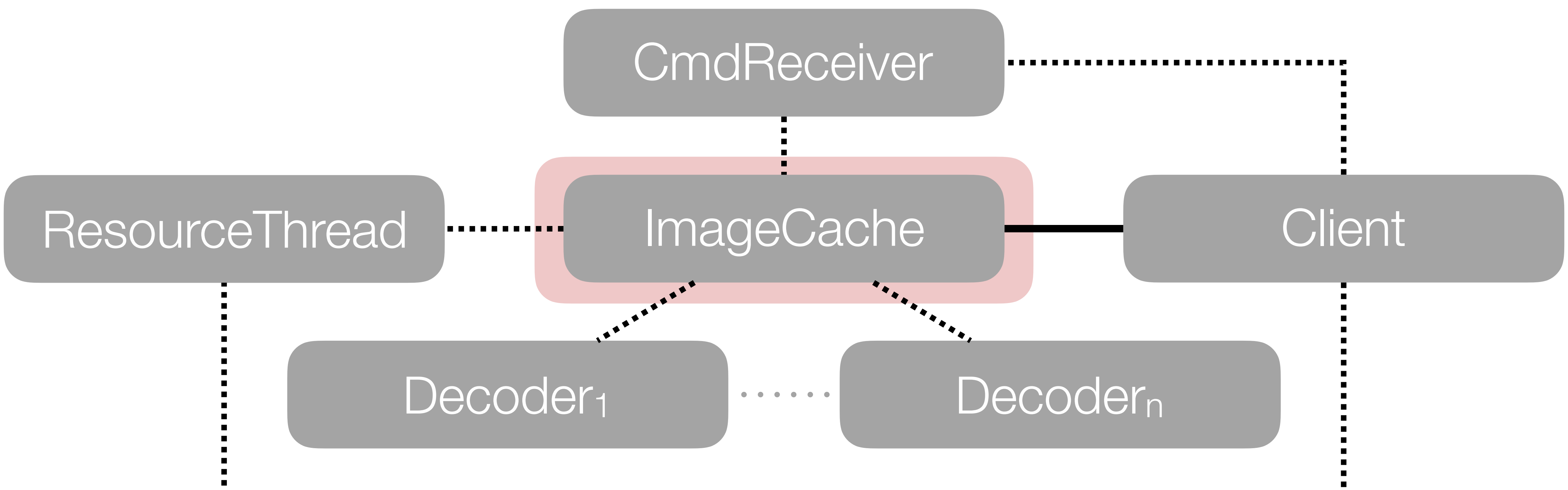


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\{\mathsf{Running} : \mathsf{ImgCacheCmd},\\ &\qquad\quad \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}\end{aligned}$$

ImageCache: $\mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$

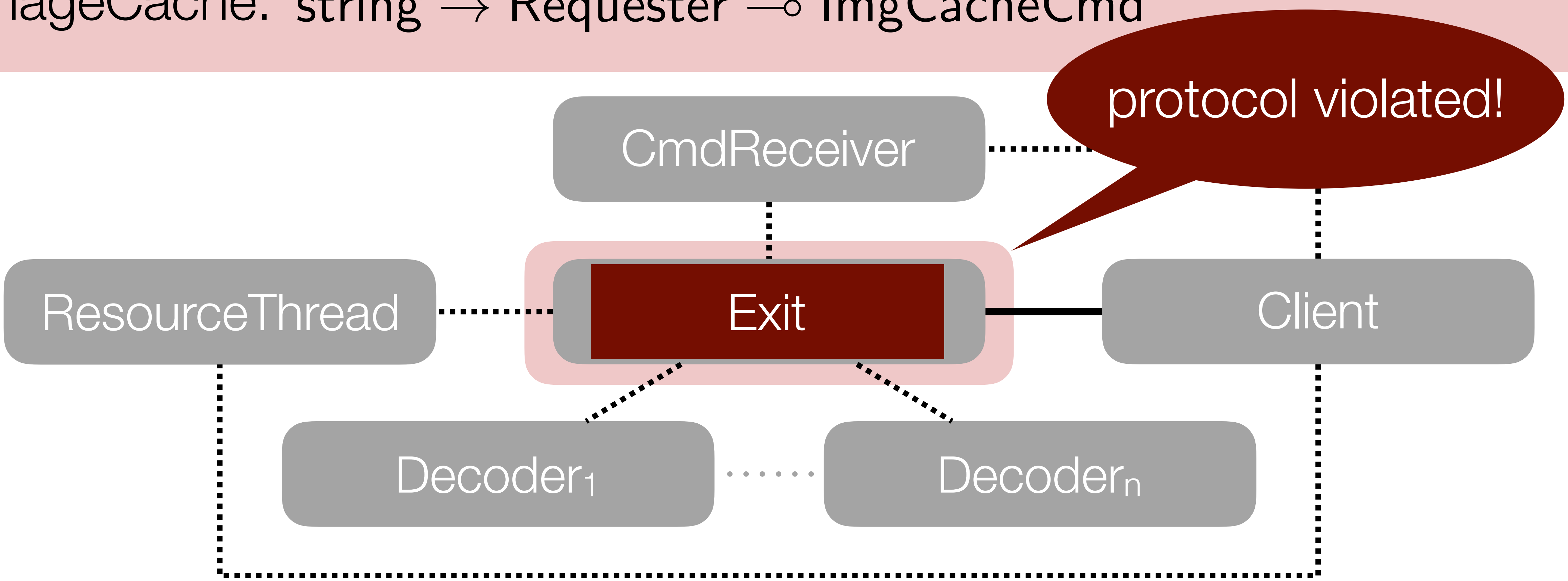


Legend: ······· shared channel —— linear channel

Have we restored protocol adherence?

$$\mathsf{ImgCacheCmd} = \&\{\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\ \mathsf{Exit} : \oplus\,\{\mathsf{Running} : \mathsf{ImgCacheCmd}, \\ \mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}\}\}$$

ImageCache: $\mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}$



Legend: ······· shared channel —— linear channel

Idea: equi-synchronizing

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\ &\mathsf{Exit} : \oplus\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}, \\ &\qquad\qquad \mathsf{Running} : \mathsf{ImgCacheCmd}\}\}\end{aligned}$$

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \overset{\text{acquire}}{\downarrow}\!\!\& \{ &\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\
&\mathsf{Exit} : \oplus \{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}, \\
&\qquad\qquad \mathsf{Running} : \mathsf{ImgCacheCmd}\}\}
\end{aligned}
$$

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

$$\mathsf{ImgCacheCmd} = \underset{\uparrow}{\overset{\text{acquire}}{}}\&\{\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \overset{\text{release}}{\downarrow}\ \mathsf{ImgCacheCmd},\\ \mathsf{Exit} : \oplus\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},\\ \mathsf{Running} : \mathsf{ImgCacheCmd}\}\}$$

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

acquire ... release ✓

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},\\ &\qquad\quad \mathsf{Running} : \mathsf{ImgCacheCmd}\}\}\end{aligned}$$

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

acquire ... release ✓

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},\\ &\qquad\mathsf{Running} : \mathsf{ImgCacheCmd}\}\}\end{aligned}$$

release

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

$$\mathsf{ImgCacheCmd} = \overset{\text{acquire}}{\downarrow}\&\{\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \overset{\text{release ✓}}{\downarrow}\mathsf{ImgCacheCmd},$$
$$\mathsf{Exit} : \oplus\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},$$
$$\mathsf{Running} : \underset{\text{release ✓}}{\uparrow}\mathsf{ImgCacheCmd}\}\}$$

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

acquire release release ✓

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\,\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},\\ &\qquad\mathsf{Running} : \mathsf{ImgCacheCmd}\}\}\end{aligned}$$

release ✓

Idea: equi-synchronizing

In addition to imposing acquire-release on shared channels, shared channels must be equi-synchronizing:

i.e., shared channels must be released back to the same type at which they were acquired, if released.

acquire ↓ release ✗ ↓ release ✓ ↓

$$\begin{aligned}\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\ &\mathsf{Exit} : \oplus\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},\\ &\qquad\quad \mathsf{Running} : \mathsf{ImgCacheCmd}\}\}\end{aligned}$$

↑ release ✓

Taking stock

Taking stock

Acquire-release + equi-synchronizing:

Taking stock

- Acquire-release + equi-synchronizing:
  - restore protocol adherence;

Taking stock

- Acquire-release + equi-synchronizing:
  - restore protocol adherence;
  - guarantee freedom of (high-level) data races because execution between acquire-release is atomic.

Taking stock

- Acquire-release + equi-synchronizing:
  - restore protocol adherence;
  - guarantee freedom of (high-level) data races because execution between acquire-release is atomic.
- We could state the policy of acquire-release and equi-synchronizing as a programming methodology.

Taking stock

- Acquire-release + equi-synchronizing:
  - restore protocol adherence;
  - guarantee freedom of (high-level) data races because execution between acquire-release is atomic.
- We could state the policy of acquire-release and equi-synchronizing as a programming methodology.
- But, why not lift this policy to the type level and have it enforced statically?

Manifest sharing

Manifest sharing

Manifest sharing

acquire

release

$$
\begin{aligned}
\mathsf{ImgCacheCmd} = \&\{&\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd},\\
&\mathsf{Exit} : \oplus\,\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1},\\
&\qquad\qquad \mathsf{Running} : \mathsf{ImgCacheCmd}\}\}
\end{aligned}
$$

release

Manifest sharing

$$\begin{array}{l} \text{ImgCacheCmd} = \underset{\text{acquire}}{\&}\{\text{RequestImage} : \text{string} \to \text{Requester} \multimap \underset{\text{release}}{\text{ImgCacheCmd}}, \\ \qquad\qquad\qquad \text{Exit} : \oplus\{\text{Done} : \text{ResourceThread} \otimes \mathbf{1}, \\ \qquad\qquad\qquad\qquad\qquad \text{Running} : \underset{\text{release}}{\text{ImgCacheCmd}}\}\} \end{array}$$

Legend: shared phase linear phase

Manifest sharing

Manifest sharing

Stratify session types into a linear and shared layer

Manifest sharing

Stratify session types into a linear and shared layer

$$A_{\mathsf{S}} \triangleq$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq \; & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \end{aligned}$$

Manifest sharing

Stratify session types into a linear and shared layer

Connect layers with modalities going back and forth

$$A_{\mathsf{S}} \triangleq$$

$$A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}}$$

Manifest sharing

- Stratify session types into a linear and shared layer
- Connect layers with modalities going back and forth

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}}
\end{array}
$$

Manifest sharing

Stratify session types into a linear and shared layer

Connect layers with modalities going back and forth

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \end{aligned}$$

Support communication of shared channels

Manifest sharing

Stratify session types into a linear and shared layer

Connect layers with modalities going back and forth

$$A_{\mathsf{S}} \triangleq \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}$$

$$\begin{array}{lll} A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{array}$$

Support communication of shared channels

Manifest sharing

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

Manifest sharing

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

$$\begin{aligned} \mathsf{ImgCacheCmd} = \ & \&\{\mathsf{RequestImage} : \mathsf{string} \rightarrow \mathsf{Requester} \multimap \mathsf{ImgCacheCmd}, \\ & \quad \mathsf{Exit} : \oplus \{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}, \\ & \qquad \mathsf{Running} : \mathsf{ImgCacheCmd}\}\} \end{aligned}$$

Manifest sharing

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\; & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

$$\begin{aligned} \mathsf{ImgCacheCmd} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{ & \mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \quad \mathsf{ImgCacheCmd}, \\ & \mathsf{Exit} : \oplus \{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}, \\ & \qquad \mathsf{Running} : \quad \mathsf{ImgCacheCmd}\}\} \end{aligned}$$

Manifest sharing

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{array}{lcl} A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{array}$$

$$\begin{array}{l} \mathsf{ImgCacheCmd} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{ImgCacheCmd}, \\ \qquad \mathsf{Exit} : \oplus \{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}, \\ \qquad\qquad \mathsf{Running} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{ImgCacheCmd}\}\} \end{array}$$

Manifest sharing

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

$$\begin{aligned} \mathsf{ImgCacheCmd} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{ & \mathsf{RequestImage} : \mathsf{string} \to \mathsf{Requester} \multimap \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{ImgCacheCmd}, \\ & \mathsf{Exit} : \oplus\{\mathsf{Done} : \mathsf{ResourceThread} \otimes \mathbf{1}, \\ & \qquad \mathsf{Running} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{ImgCacheCmd}\}\} \end{aligned}$$

Up and down shifts denote acquire and release, resp.

Typing judgments

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

Typing judgments

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

Typing judgments

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

$$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

Typing judgments

A_{S}	$\triangleq$	$\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$
$A_{\mathsf{L}}, B_{\mathsf{L}}$	$\triangleq$	$\oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid$
		$\&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}$

Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

$$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

"Process P provides session of type A_m along x_m using channels in (Γ and) Δ."

Typing judgments

A_{S}	$\triangleq$	$\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$
$A_{\mathsf{L}}, B_{\mathsf{L}}$	$\triangleq$	$\oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid$
		$\&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}$

Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

shared (structural) context

$$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

"Process P provides session of type A_m along x_m using channels in (Γ and) Δ."

Typing judgments

A_{S}	$\triangleq$	$\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$
$A_{\mathsf{L}}, B_{\mathsf{L}}$	$\triangleq$	$\oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid$
		$\&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}$

Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

$$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

"Process P provides session of type A_m along x_m using channels in (Γ and) Δ."

Typing judgments

$$A_{\mathsf{S}} \triangleq {\uparrow}^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid {\downarrow}^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

linear (substructural) context

$$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma; \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

"Process P provides session of type A_m along x_m using channels in (Γ and) Δ."

Typing of acquire

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}{\uparrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{L})$$

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\mathrm{L})$$

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})} \ (\text{T-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\text{L})$$

$$\frac{\Gamma;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})} \ (\text{T-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\text{R})$$

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\text{L})$$

$$\frac{\Gamma;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}\ (\text{T-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\text{R})$$

Typing of release

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}\downarrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})} \ (\mathrm{T}\text{-}\downarrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}\downarrow_{\mathsf{L}}^{\mathsf{S}}\mathrm{L})$$

$$\frac{\Gamma \vdash_\Sigma P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{detach}\ x_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}})}\ (\mathrm{T}\text{-}\downarrow_{\mathsf{L}}^{\mathsf{S}}\mathrm{R})$$

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})} \ (\mathrm{T}\text{-}{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

$$\frac{\Gamma \vdash_{\Sigma} P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{detach}\ x_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}})} \ (\mathrm{T}\text{-}{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathrm{R})$$

Taking stock

Taking stock

- We have a session type system that allows shared and linear channels to coexist and guarantees:
  - data-race-freedom (low-level and high-level)
  - protocol adherence

Taking stock

- We have a session type system that allows shared and linear channels to coexist and guarantees:
  - data-race-freedom (low-level and high-level)
  - protocol adherence
- What about deadlock-freedom?

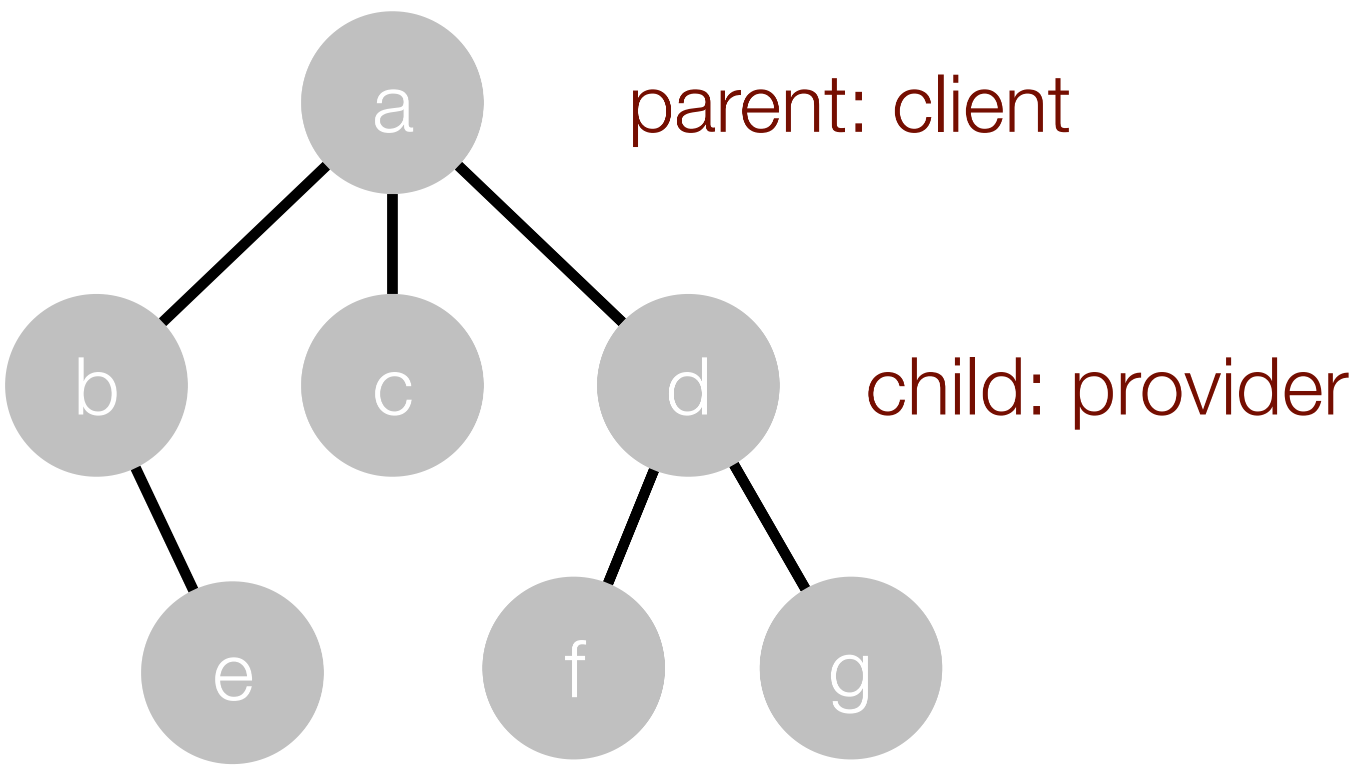
Why are linear session types deadlock-free?

Why are linear session types deadlock-free?

Linearity (“exactly one client”) turns process graph into a tree.

Why are linear session types deadlock-free?

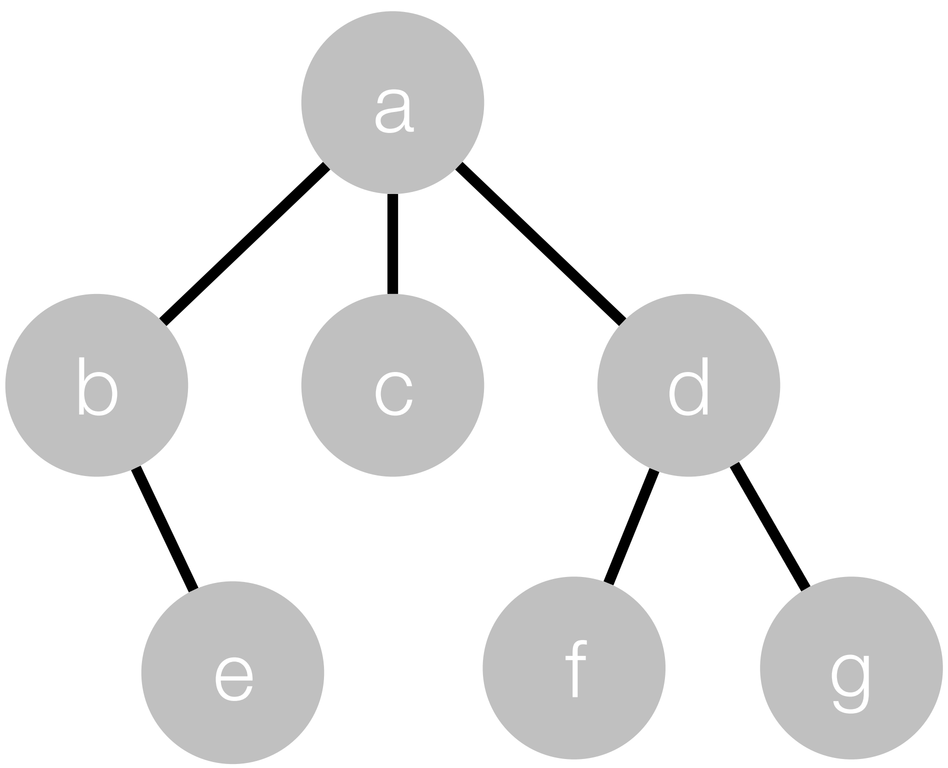
Linearity ("exactly one client") turns process graph into a tree.



Legend: —— linear channel

Why are linear session types deadlock-free?

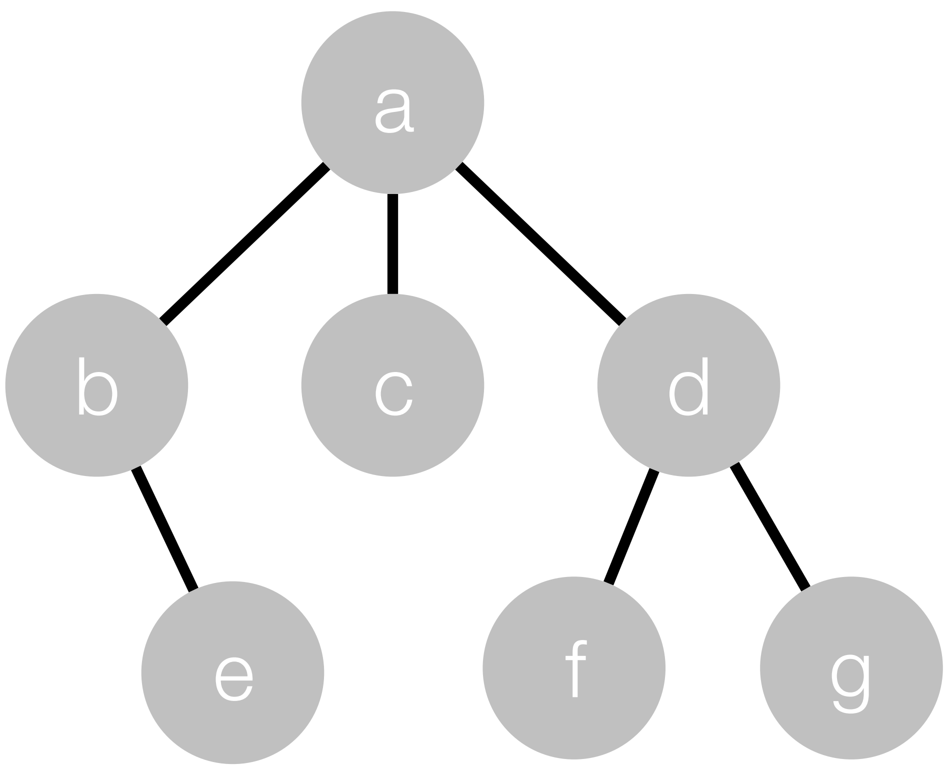
Linearity (“exactly one client”) turns process graph into a tree.



Legend: linear channel

Why are linear session types deadlock-free?

Linearity ("exactly one client") turns process graph into a tree.

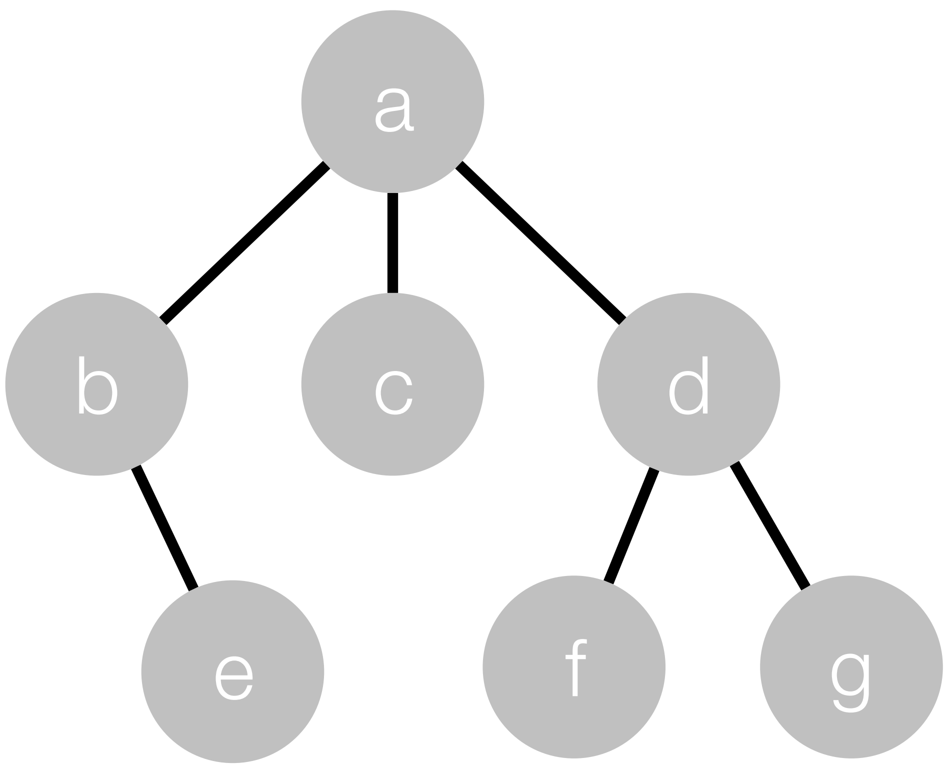


What are the threats to progress?

Legend: —— linear channel

Why are linear session types deadlock-free?

Linearity (“exactly one client”) turns process graph into a tree.



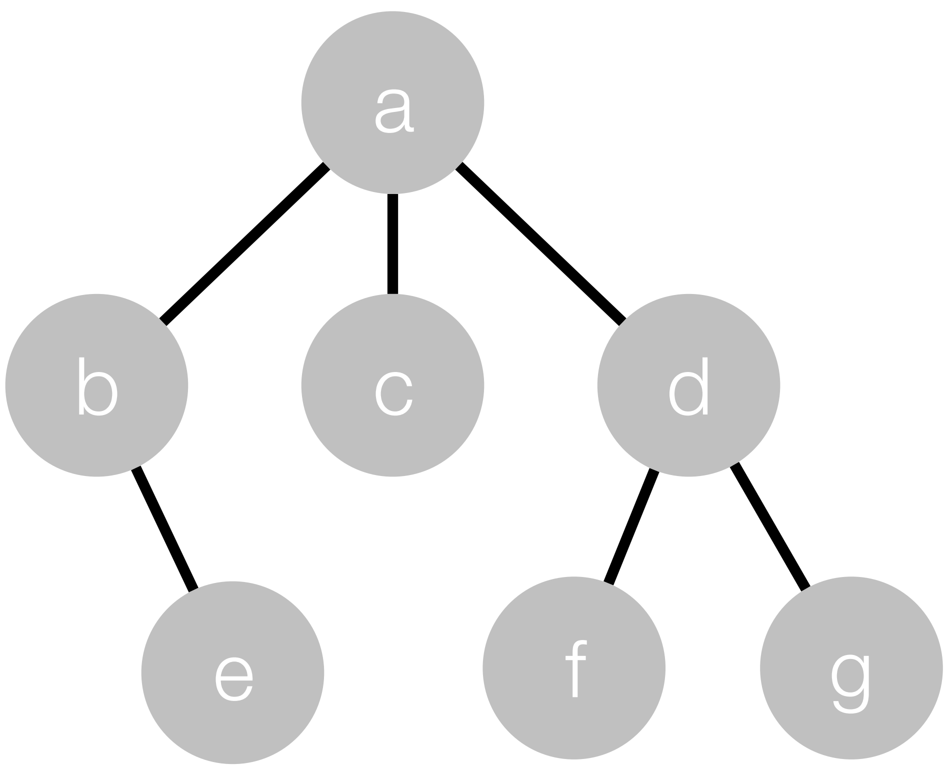
What are the threats to progress?

- Two scenarios:

Legend: —— linear channel

Why are linear session types deadlock-free?

Linearity ("exactly one client") turns process graph into a tree.



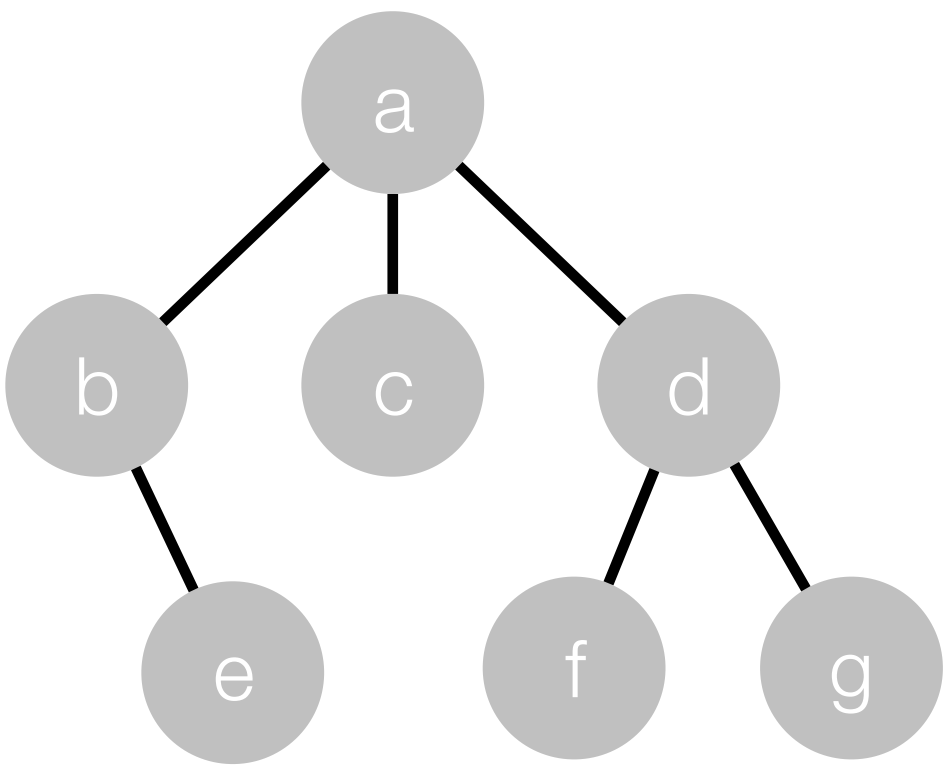
What are the threats to progress?

- Two scenarios:
  - provider ready to synchronize, client not

Legend: —— linear channel

Why are linear session types deadlock-free?

Linearity ("exactly one client") turns process graph into a tree.



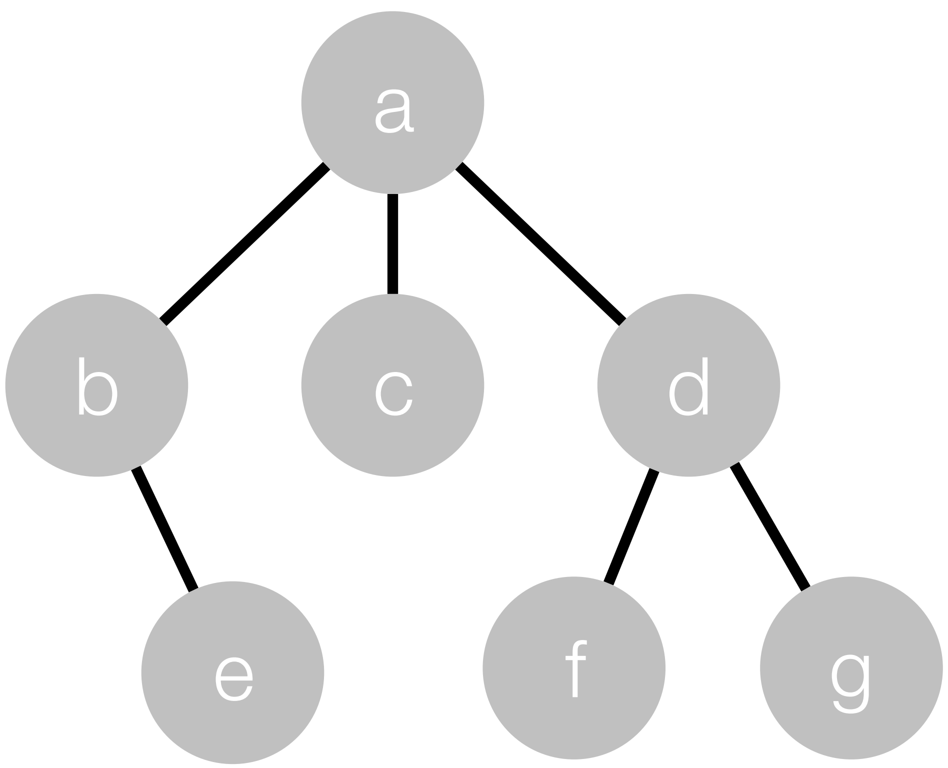
What are the threats to progress?

- Two scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not

Legend: —— linear channel

Why are linear session types deadlock-free?

Linearity ("exactly one client") turns process graph into a tree.



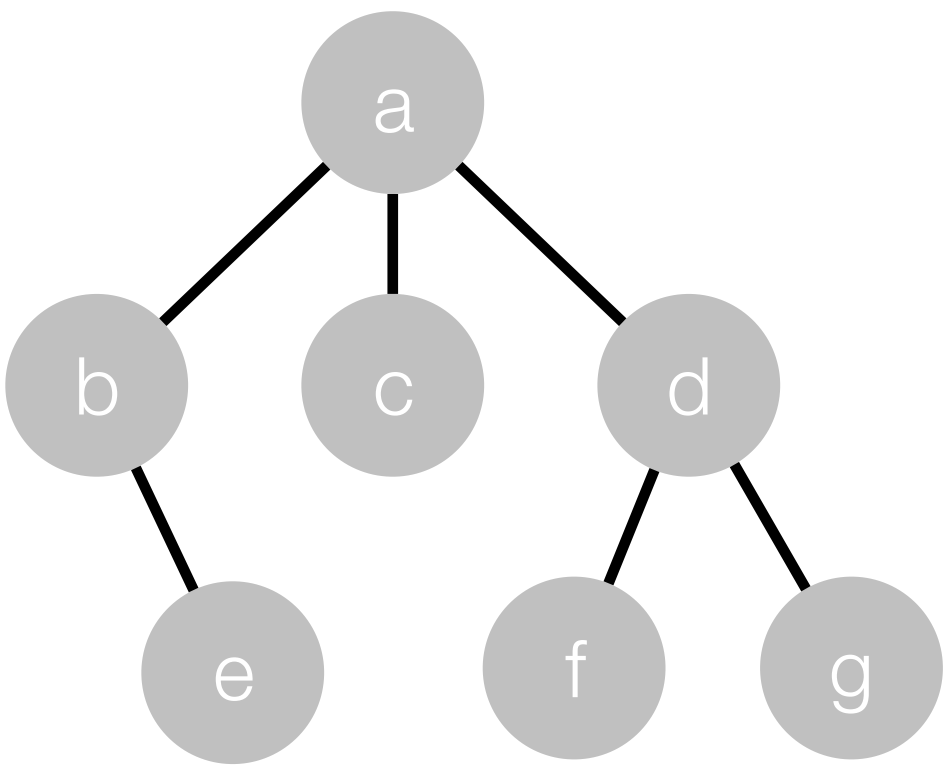
What are the threats to progress?

- Two scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not
- Let's visualize this waiting dependency with a green arrow

Legend: —— linear channel

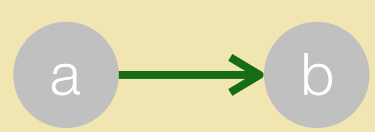
Why are linear session types deadlock-free?

Linearity ("exactly one client") turns process graph into a tree.



What are the threats to progress?

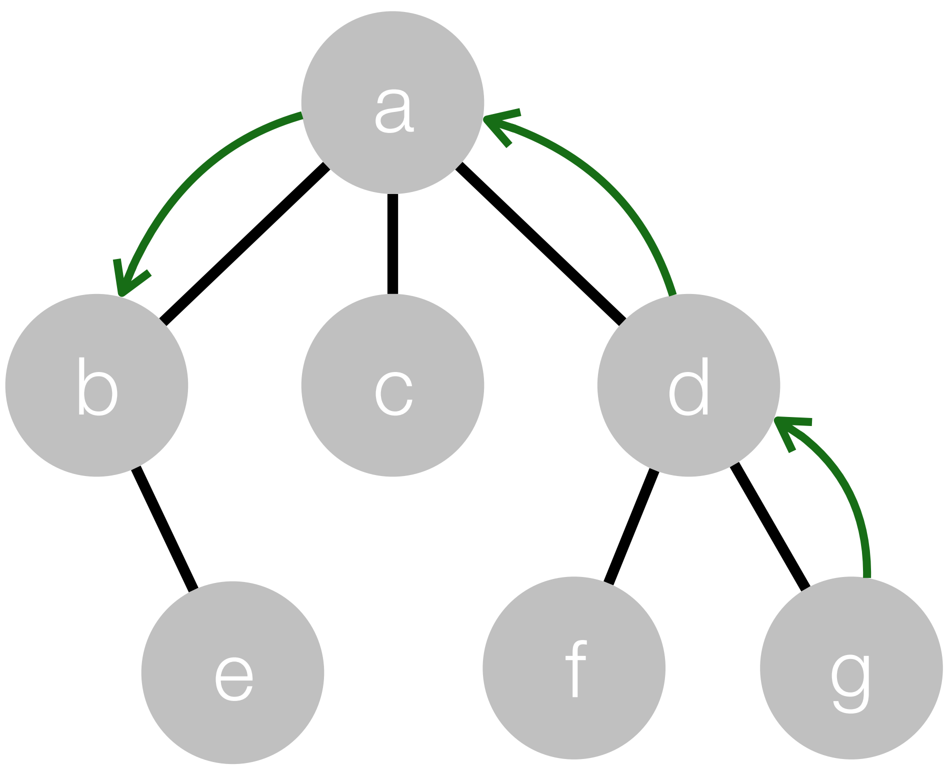
- Two scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not
- Let's visualize this waiting dependency with a green arrow

Legend: —— linear channel 

"a waits for b"

Why are linear session types deadlock-free?

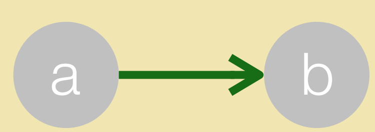
Linearity ("exactly one client") turns process graph into a tree.



What are the threats to progress?

- Two scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not
- Let's visualize this waiting dependency with a green arrow

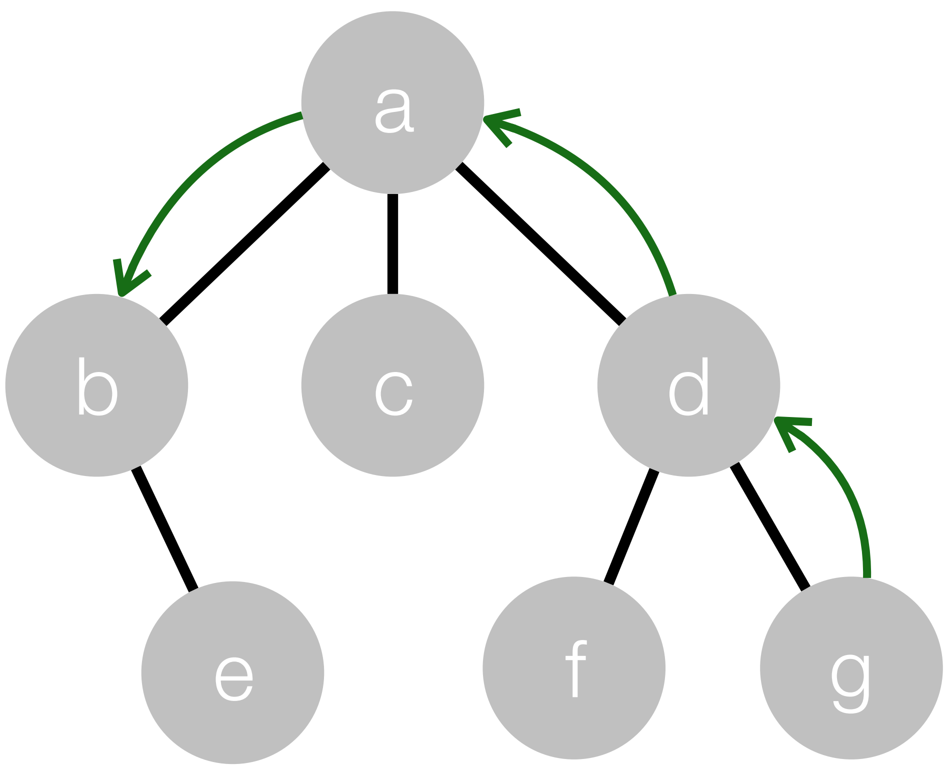
Legend: linear channel



"a waits for b"

Why are linear session types deadlock-free?

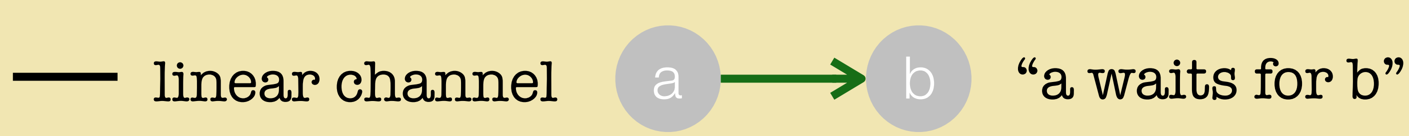
Linearity ("exactly one client") turns process graph into a tree.



What are the threats to progress?

- Two scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not
- Let's visualize this waiting dependency with a green arrow

No green cycles: green arrows can only go along linear channels, and client and provider cannot both be waiting for each other.

Legend: 

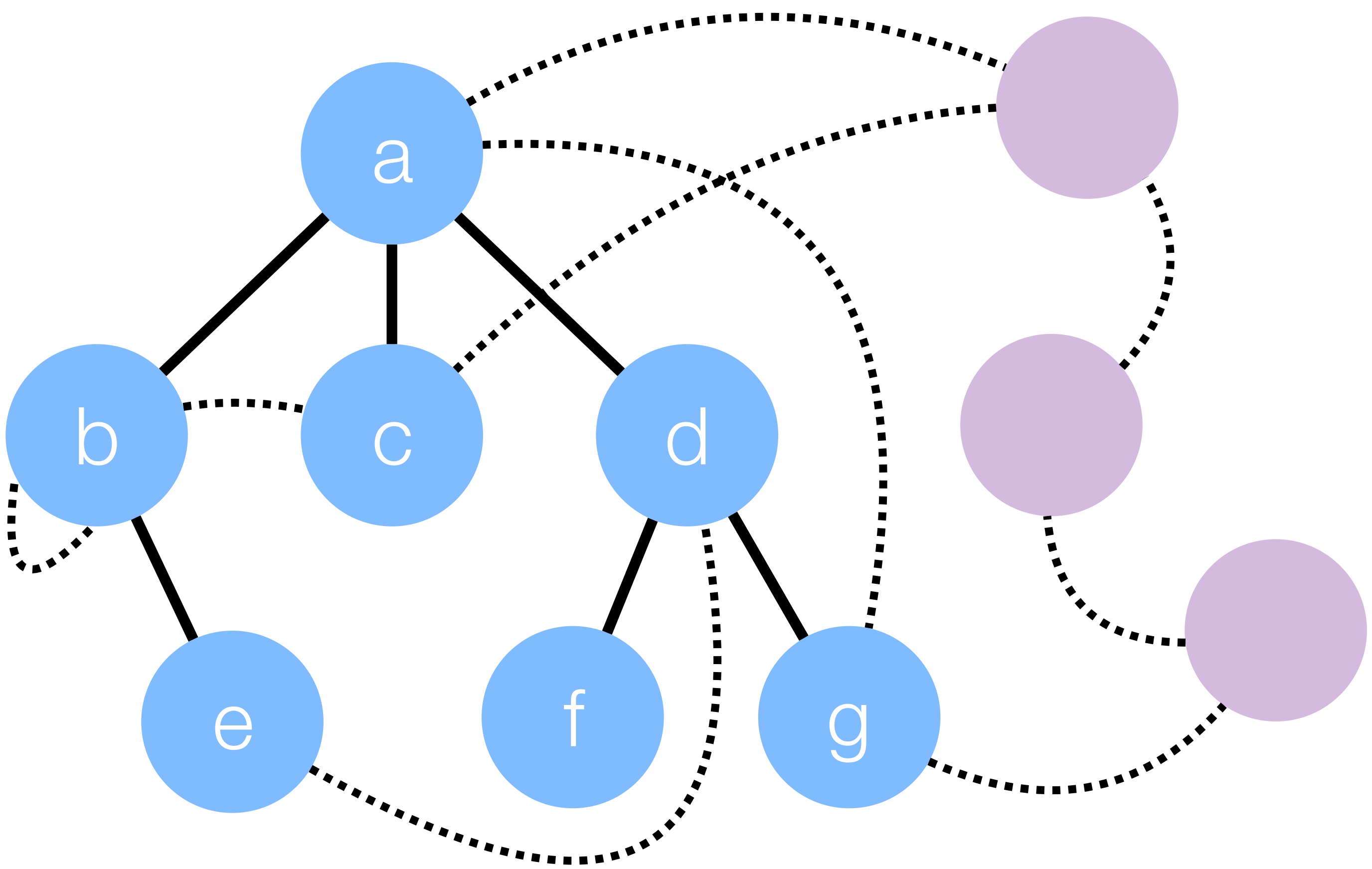
Let's add sharing

Let’s add sharing

We get a graph of linear and shared processes, with a linear tree inside.

Let’s add sharing

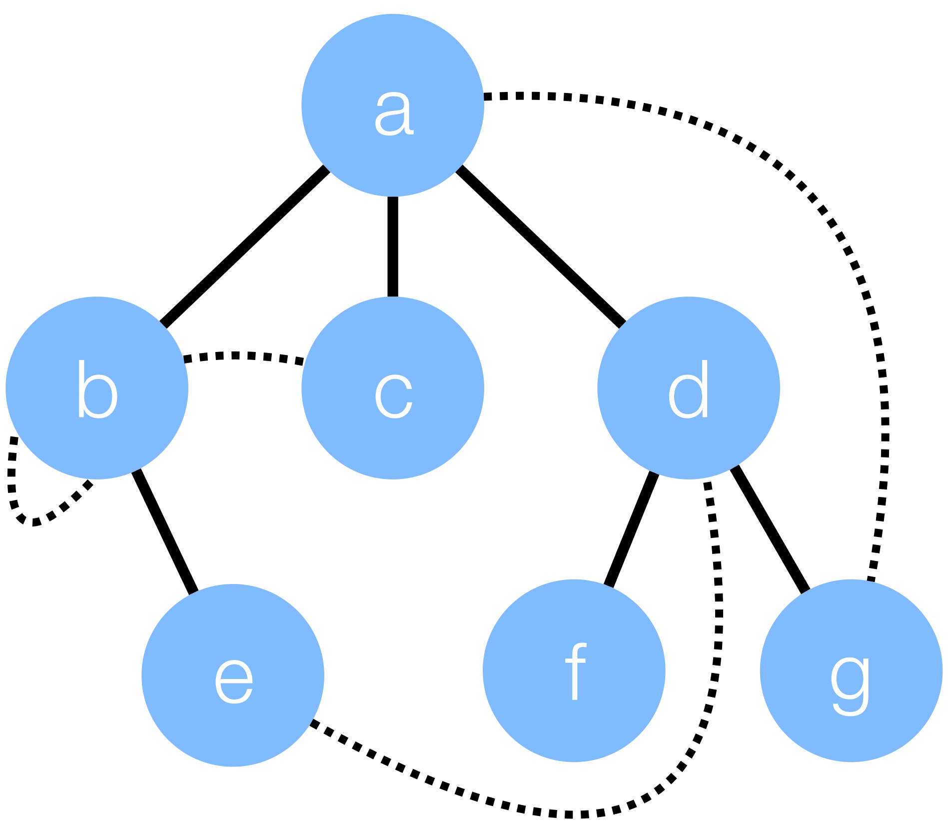
We get a graph of linear and shared processes, with a linear tree inside.



Legend: linear channel · shared channel · linear process · shared process

Let's add sharing

We get a graph of linear and shared processes, with a linear tree inside.

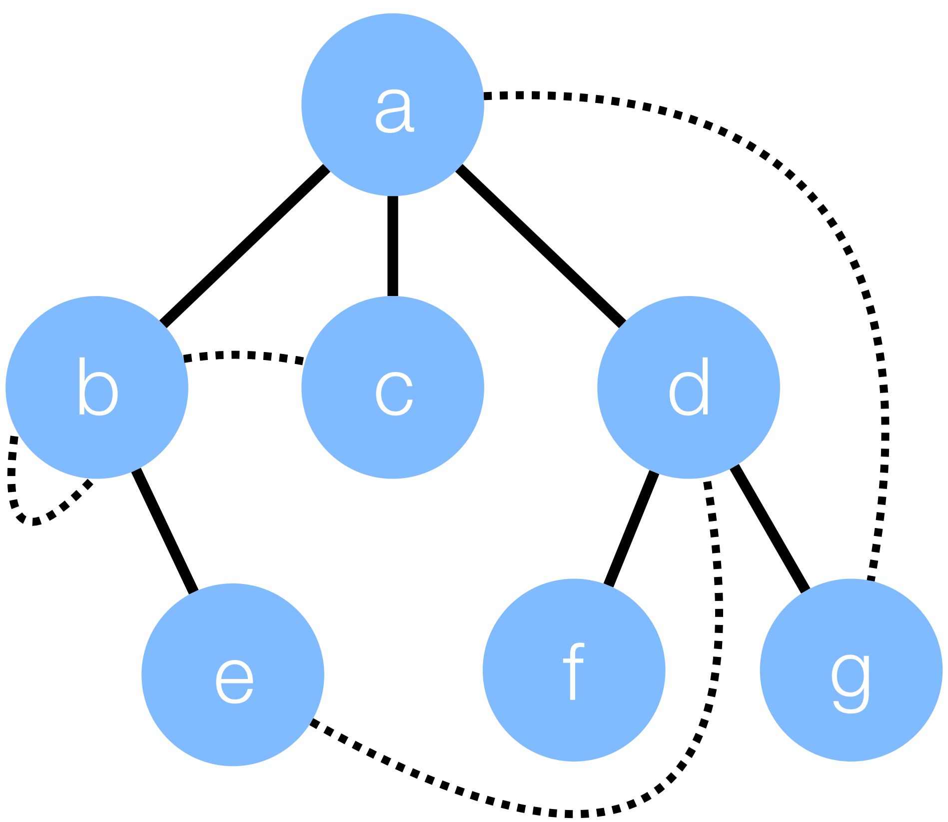


Legend: — linear channel ● linear process ● shared process

······ shared channel

Let’s add sharing

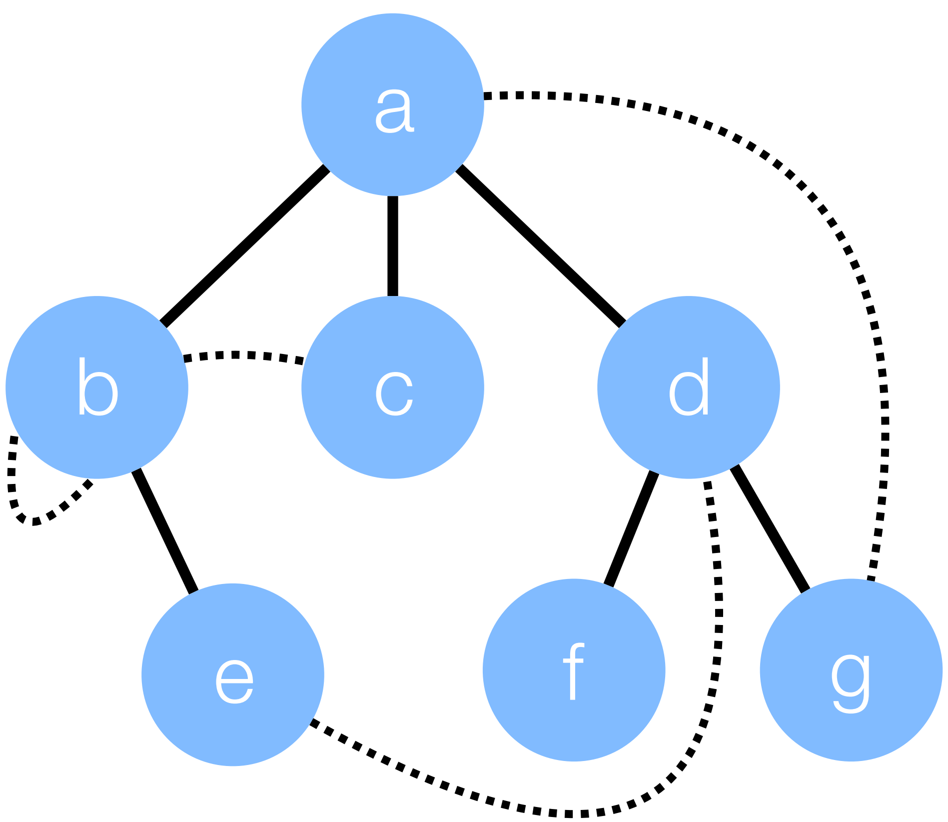
We get a graph of linear and shared processes, with a linear tree inside.



Legend: linear channel; shared channel; linear process; shared process

Let's add sharing

We get a graph of linear and shared processes, with a linear tree inside.

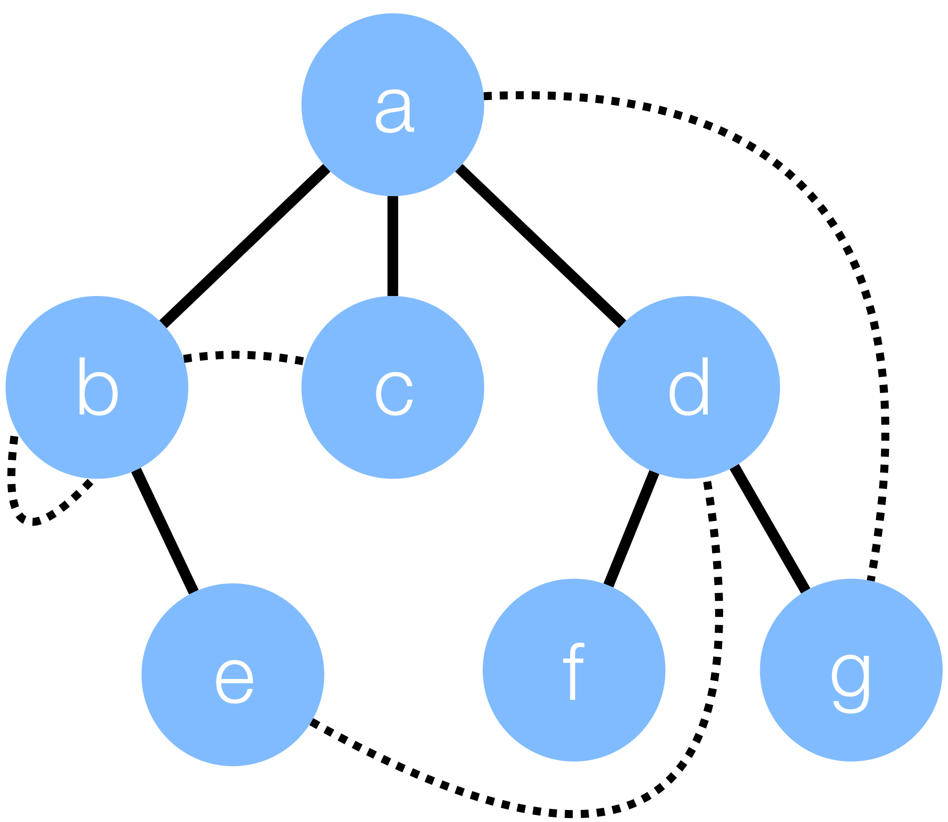


Acquire-release amounts to "locking"

Legend: linear channel · shared channel · linear process · shared process

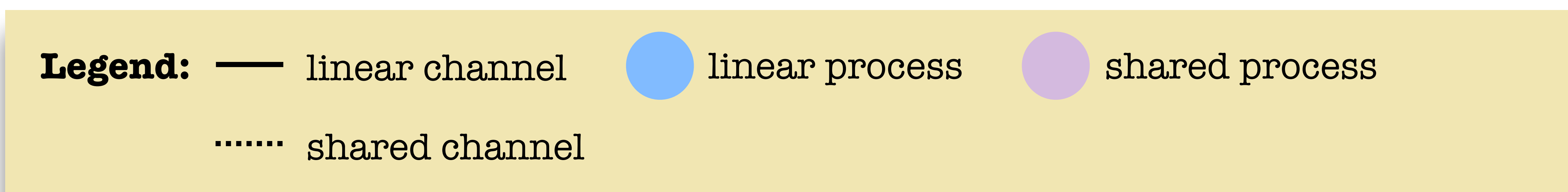
Let's add sharing

We get a graph of linear and shared processes, with a linear tree inside.



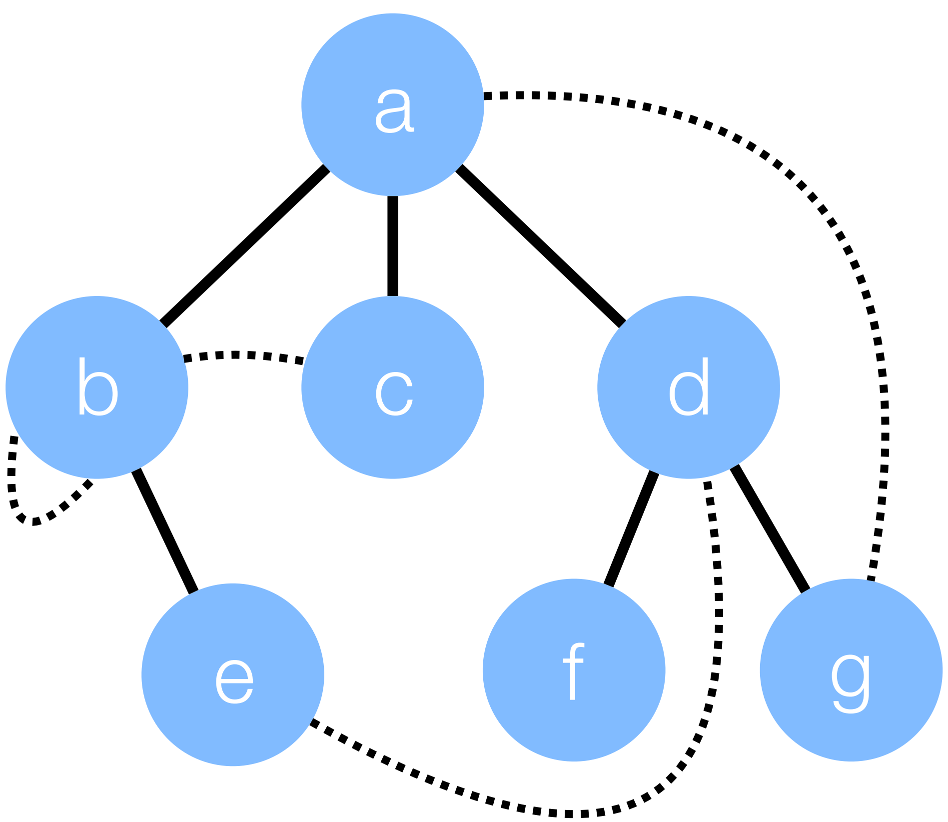
Acquire-release amounts to "locking"

- Possibility of cyclic dependencies



Let's add sharing

We get a graph of linear and shared processes, with a linear tree inside.



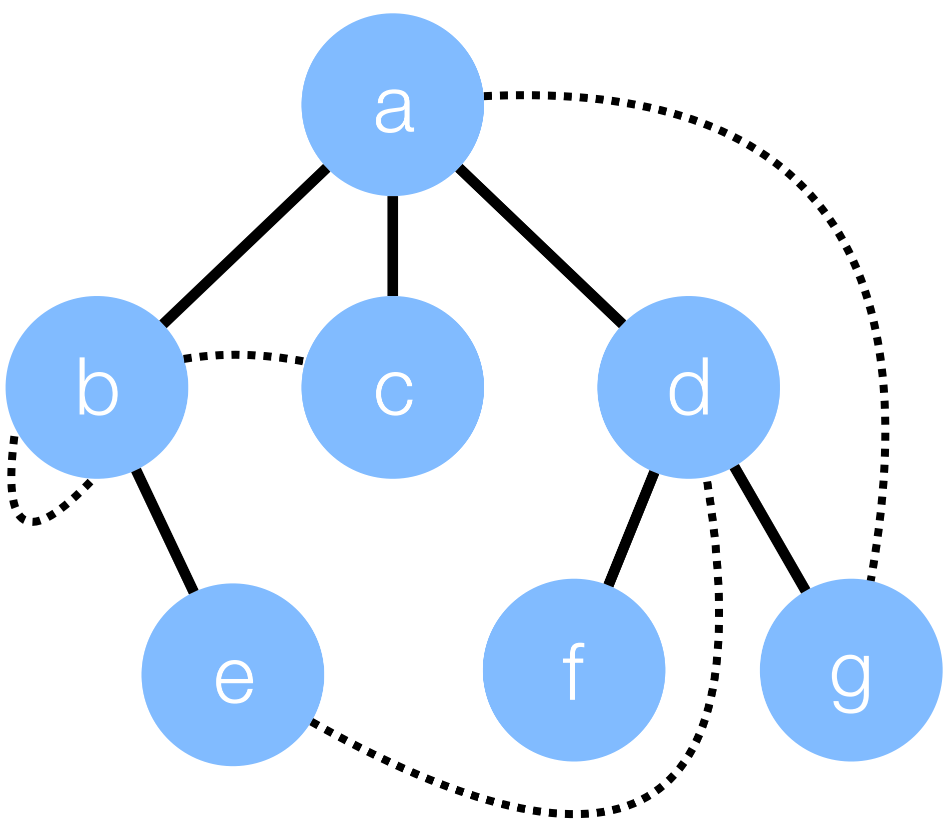
Acquire-release amounts to "locking"

- Possibility of cyclic dependencies
- Let's visualize this waiting dependency with a red arrow

Legend: linear channel · linear process · shared process · shared channel

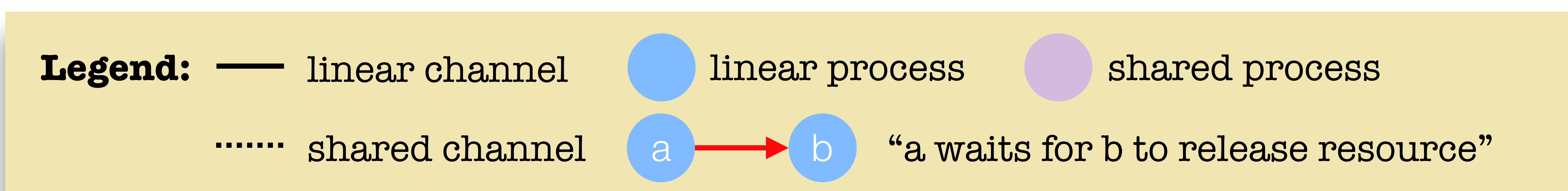
Let’s add sharing

We get a graph of linear and shared processes, with a linear tree inside.



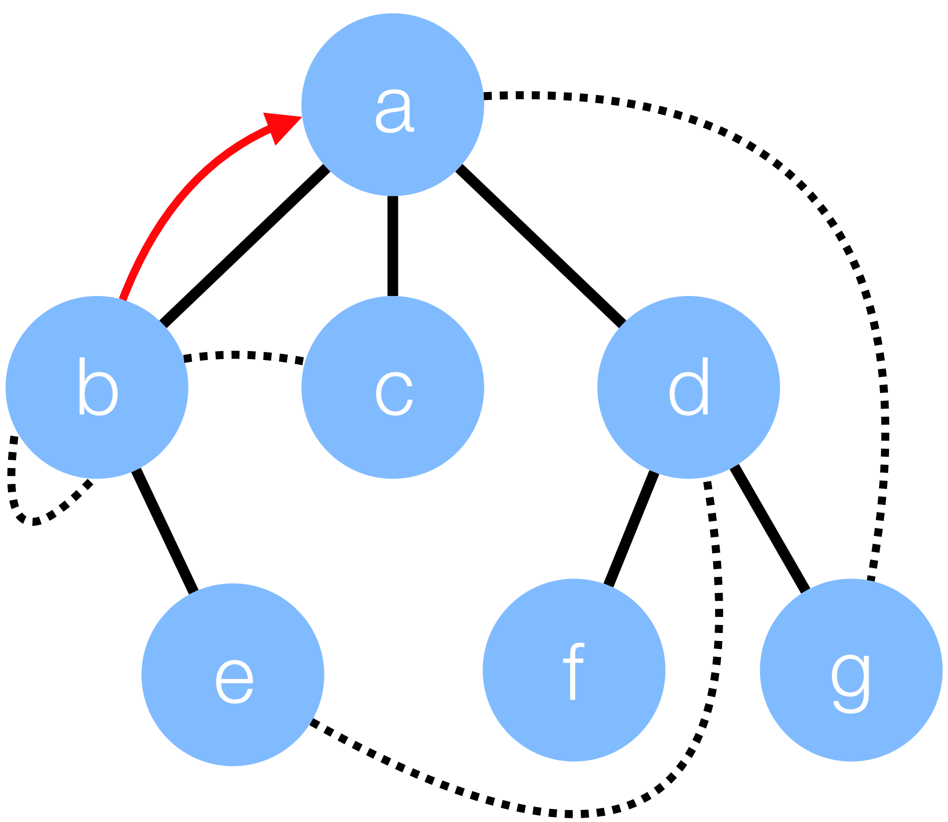
Acquire-release amounts to “locking”

- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow



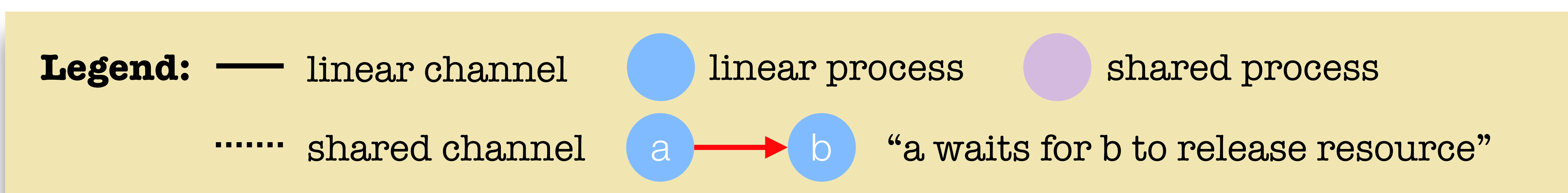
Let's add sharing

We get a graph of linear and shared processes, with a linear tree inside.



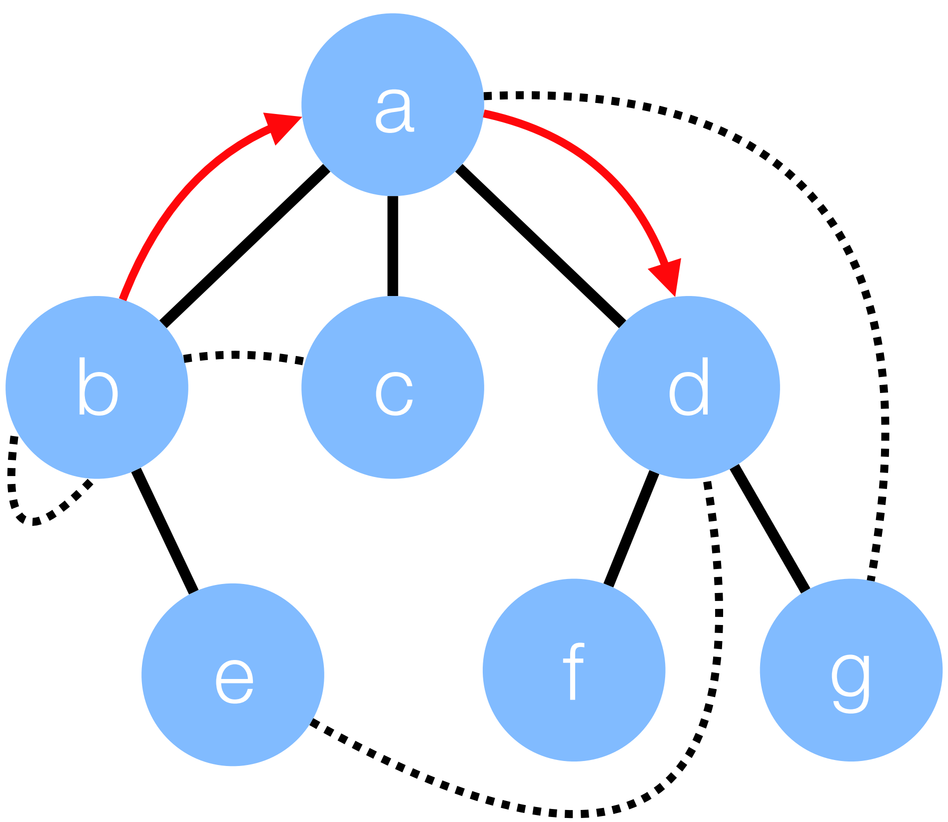
Acquire-release amounts to "locking"

- Possibility of cyclic dependencies
- Let's visualize this waiting dependency with a red arrow



Let’s add sharing

We get a graph of linear and shared processes, with a linear tree inside.



Acquire-release amounts to “locking”

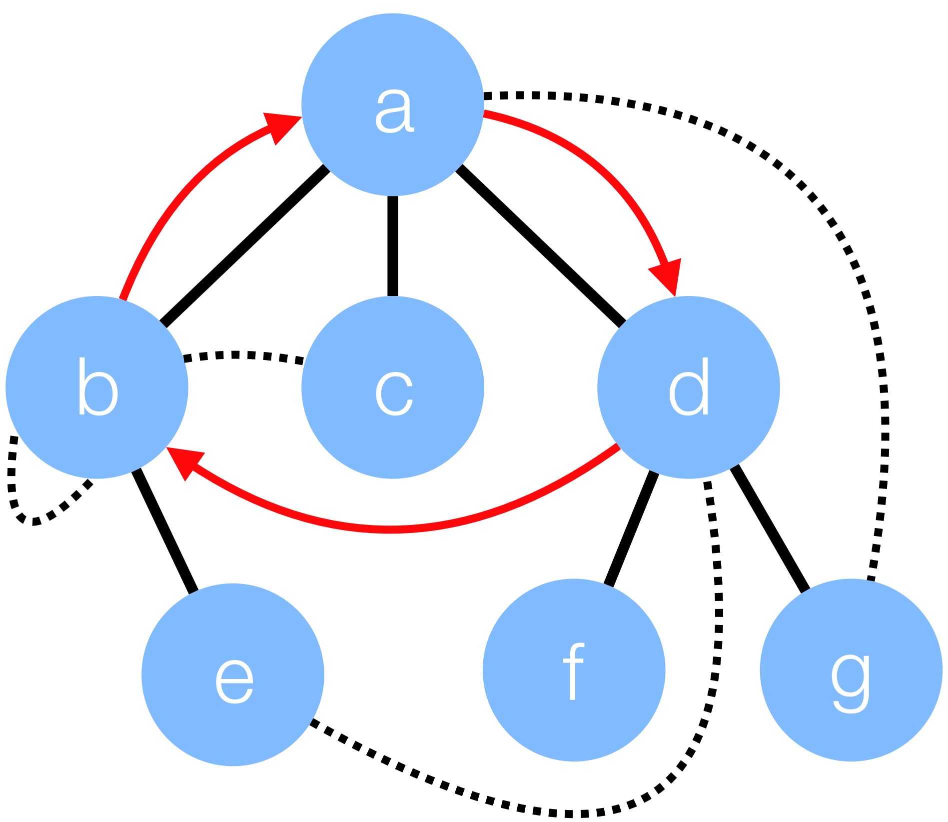
- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow

Legend: linear channel — linear process — shared process

shared channel — a → b “a waits for b to release resource”

Let’s add sharing

We get a graph of linear and shared processes, with a linear tree inside.



Acquire-release amounts to “locking”

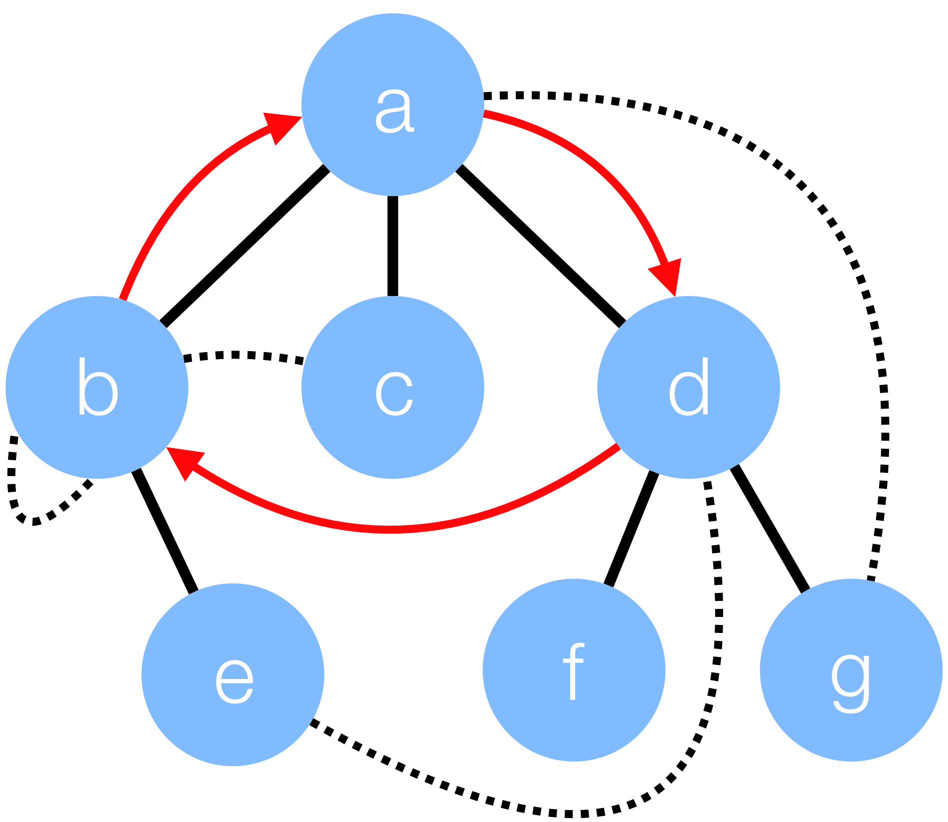
- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow

Legend: linear channel — linear process — shared process

shared channel — a → b “a waits for b to release resource”

Let's add sharing

We get a graph of linear and shared processes, with a linear tree inside.



Acquire-release amounts to "locking"

- Possibility of cyclic dependencies
- Let's visualize this waiting dependency with a red arrow
- Note: red arrows can connect arbitrary nodes

Legend: linear channel — linear process — shared process

shared channel — a → b "a waits for b to release resource"

Can we re-establish deadlock-freedom?

Can we re-establish deadlock-freedom?

An enticing solution: “locking up”

- Impose a partial order on resources.
- Ensure that resources are acquired (“locked”) in increasing order.

Can we re-establish deadlock-freedom?

An enticing solution: “locking up”

- Impose a partial order on resources.
- Ensure that resources are acquired (“locked”) in increasing order.

→ However, cyclic dependencies between acquire requests are not the only source of deadlock!

Can we re-establish deadlock-freedom?

An enticing solution: “locking up”

- Impose a partial order on resources.
- Ensure that resources are acquired (“locked”) in increasing order.

➡ However, cyclic dependencies between acquire requests are not the only source of deadlock!

Two Forms of waiting dependencies:

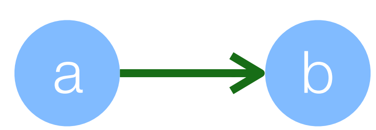
Can we re-establish deadlock-freedom?

An enticing solution: “locking up”

- Impose a partial order on resources.
- Ensure that resources are acquired (“locked”) in increasing order.

However, cyclic dependencies between acquire requests are not the only source of deadlock!

Two Forms of waiting dependencies:

- waiting to synchronize: 

“a waits for b to synchronize”

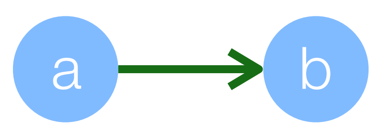
Can we re-establish deadlock-freedom?

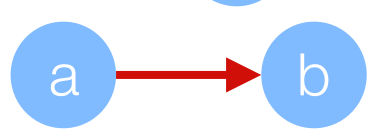
An enticing solution: “locking up”

- Impose a partial order on resources.
- Ensure that resources are acquired (“locked”) in increasing order.

→ However, cyclic dependencies between acquire requests are not the only source of deadlock!

Two Forms of waiting dependencies:

- waiting to synchronize: 

“a waits for b to synchronize”
- waiting to release: 

“a waits for b to release resource”

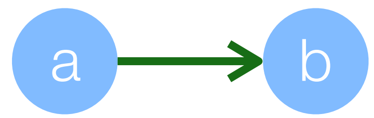
Can we re-establish deadlock-freedom?

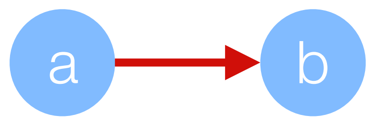
Two Forms of waiting dependencies:

- waiting to synchronize: $a \rightarrow b$ "a waits for b to synchronize"
- waiting to release: $a \rightarrow b$ "a waits for b to release resource"

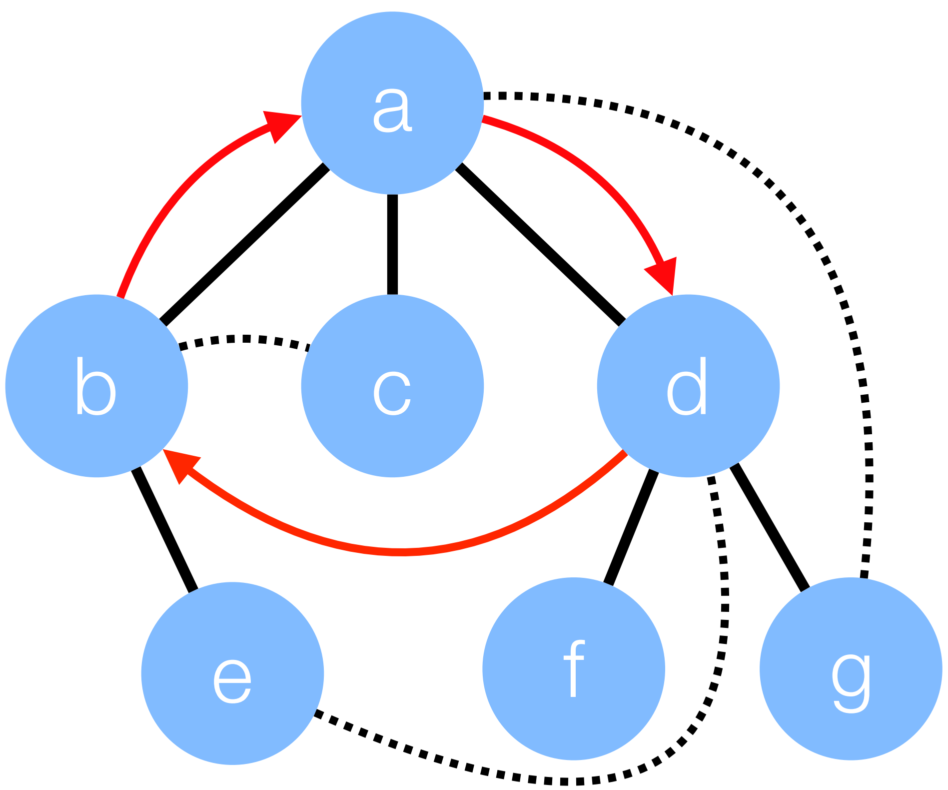
Can we re-establish deadlock-freedom?

Two Forms of waiting dependencies:

- waiting to synchronize: 

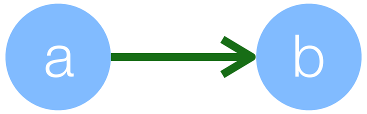
“a waits for b to synchronize”
- waiting to release: 

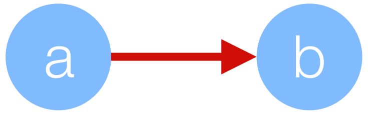
“a waits for b to release resource”



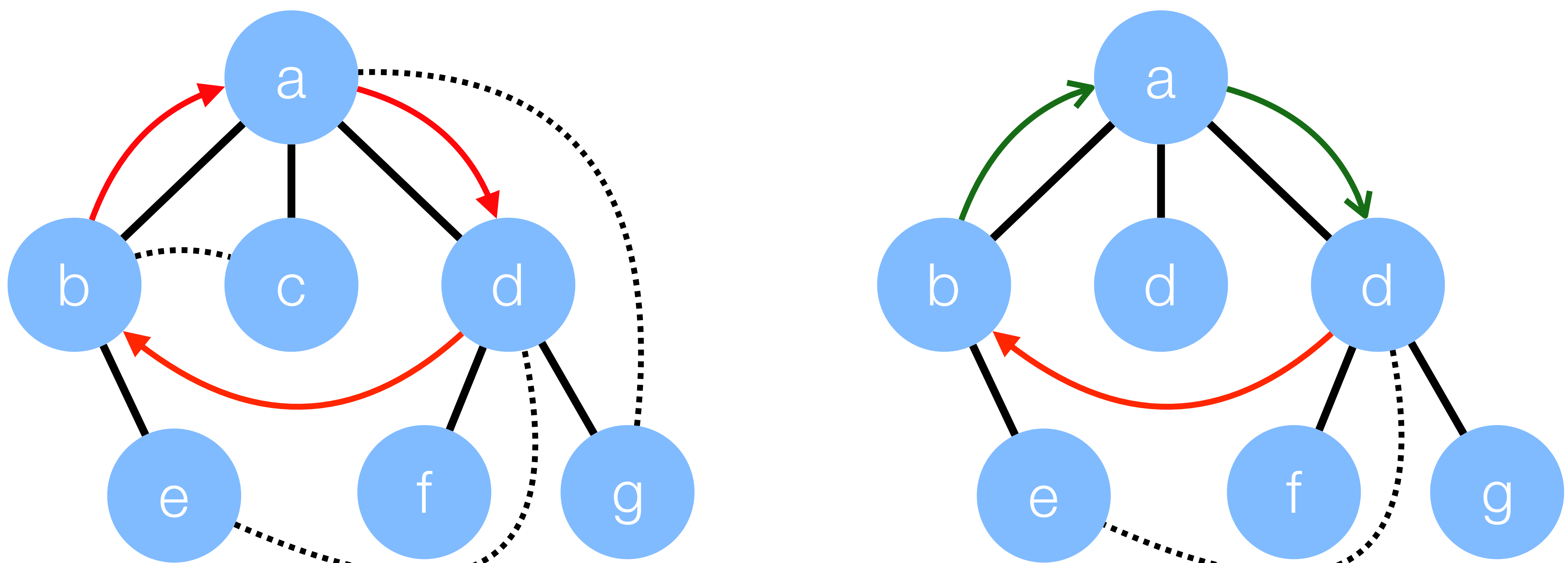
Can we re-establish deadlock-freedom?

Two Forms of waiting dependencies:

- waiting to synchronize: 

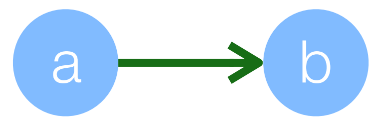
“a waits for b to synchronize”
- waiting to release: 

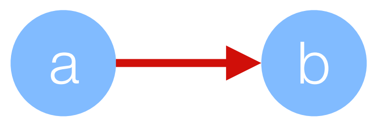
“a waits for b to release resource”



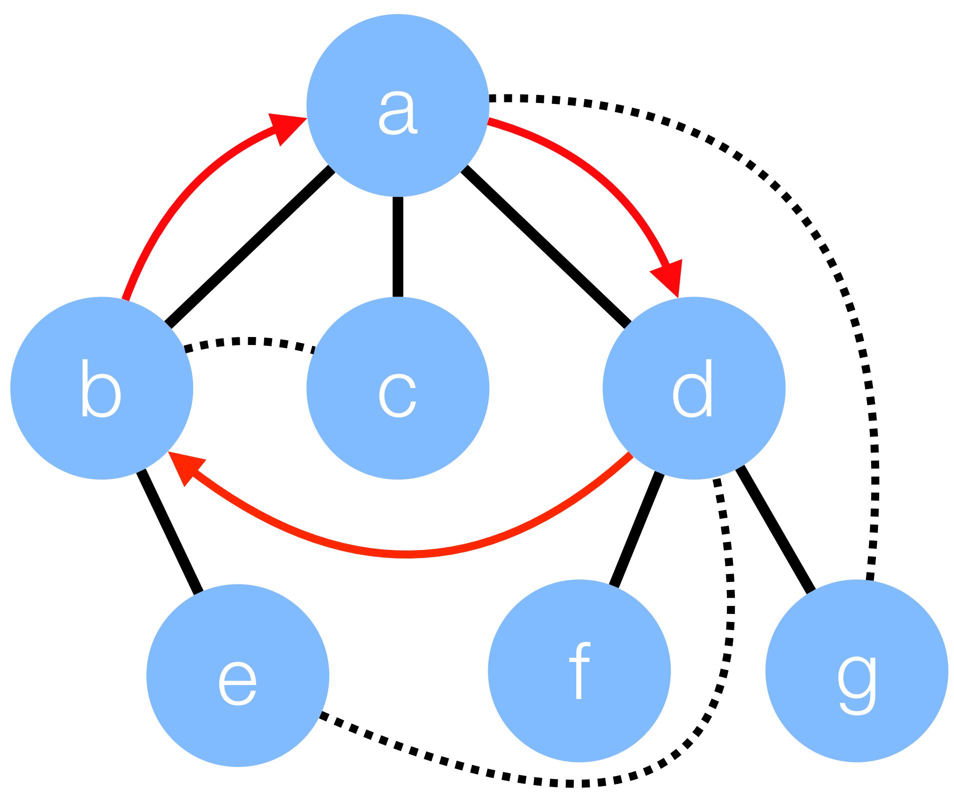
Can we re-establish deadlock-freedom?

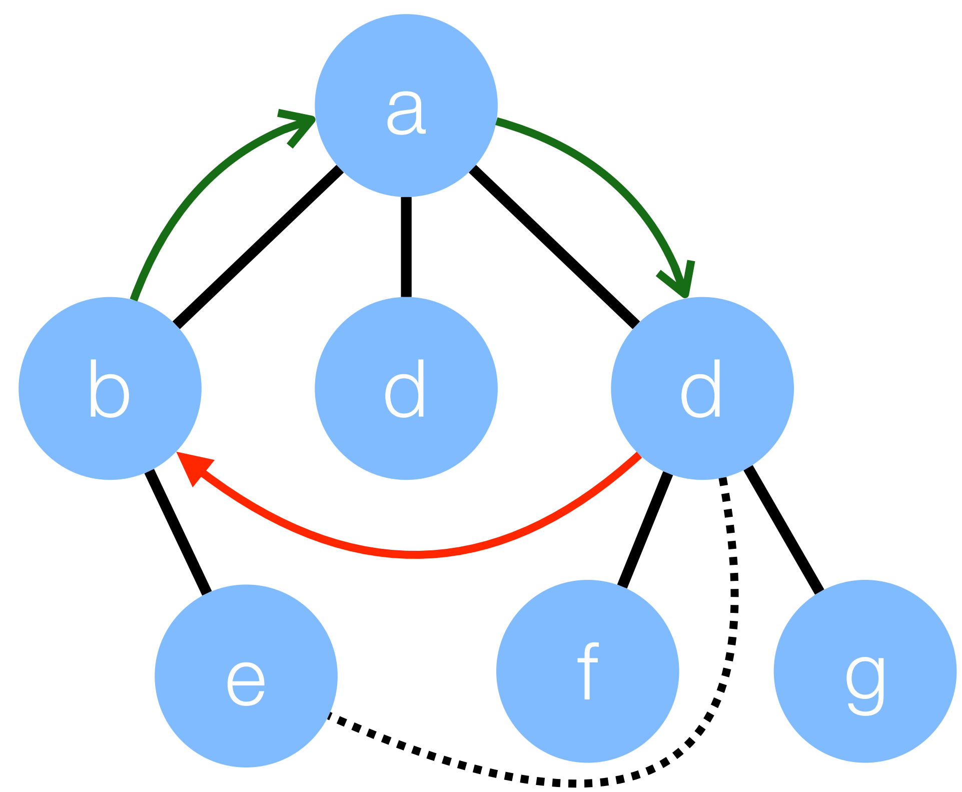
Two Forms of waiting dependencies:

- waiting to synchronize: 

“a waits for b to synchronize”
- waiting to release: 

“a waits for b to release resource”





Cycles can consist of red arrows only or a combination of red and green arrows.

Idea: competitors and collaborators

Idea: competitors and collaborators

Competitors: overlap in set of resources acquired

Idea: competitors and collaborators

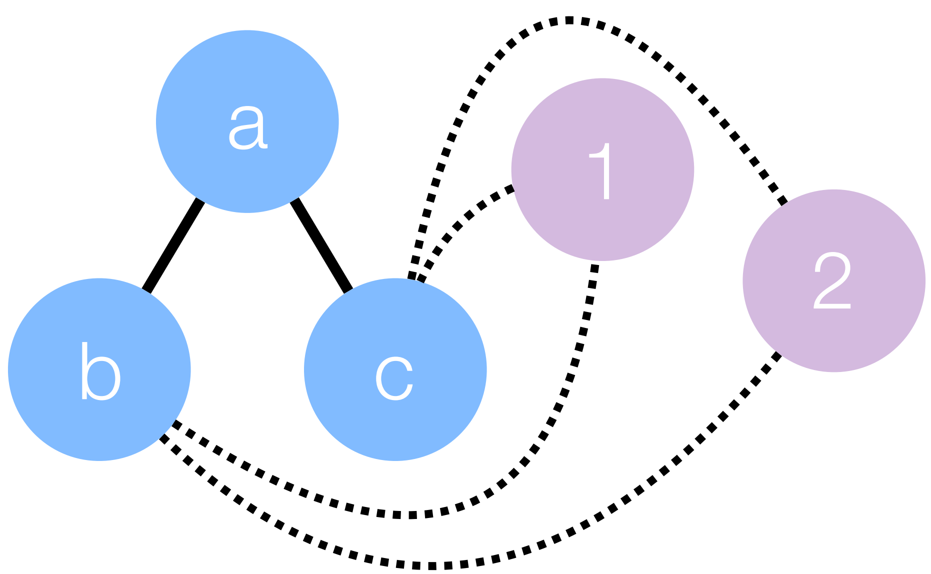
- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired

Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer

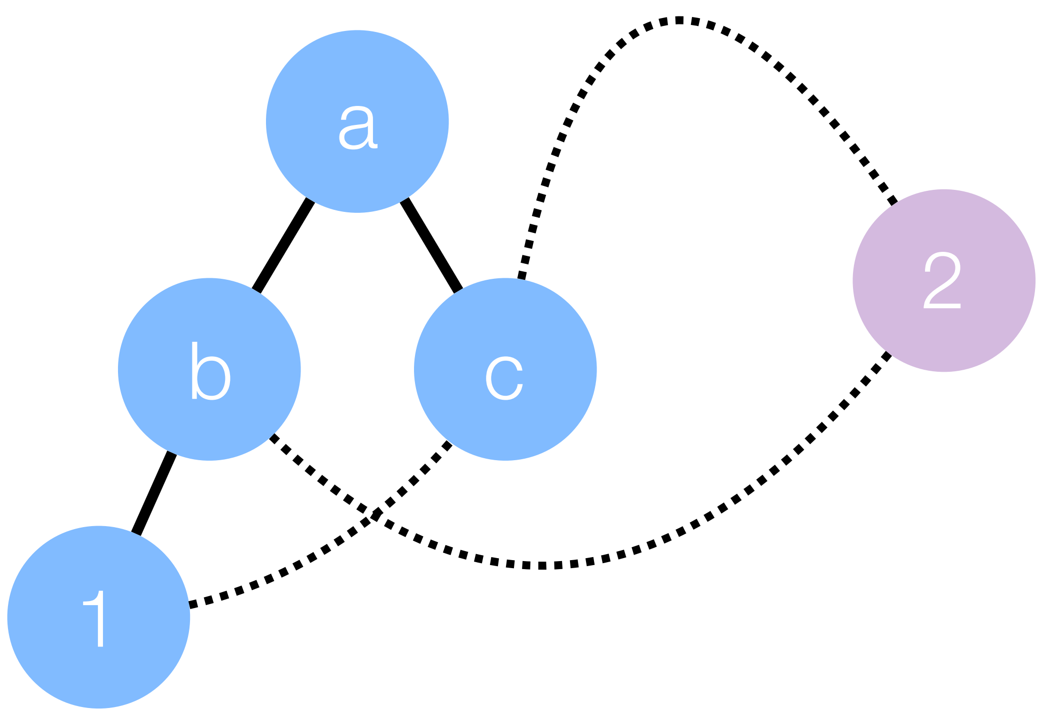
Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer



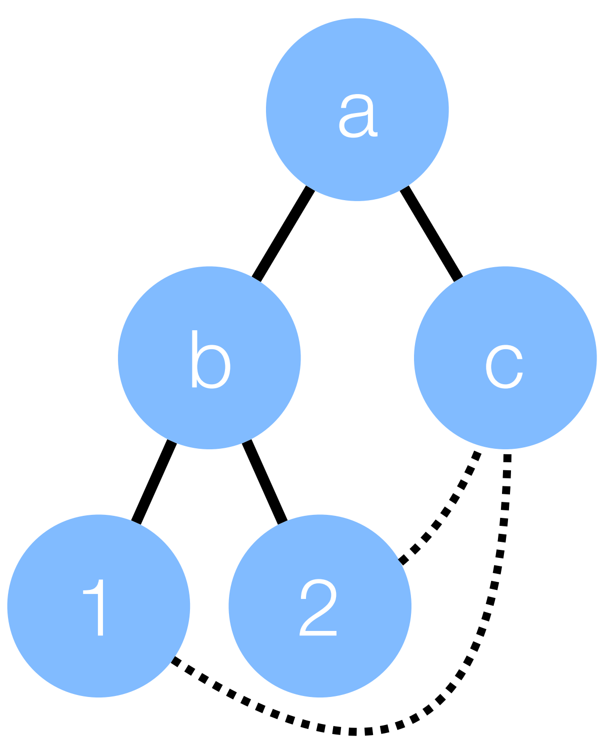
Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer



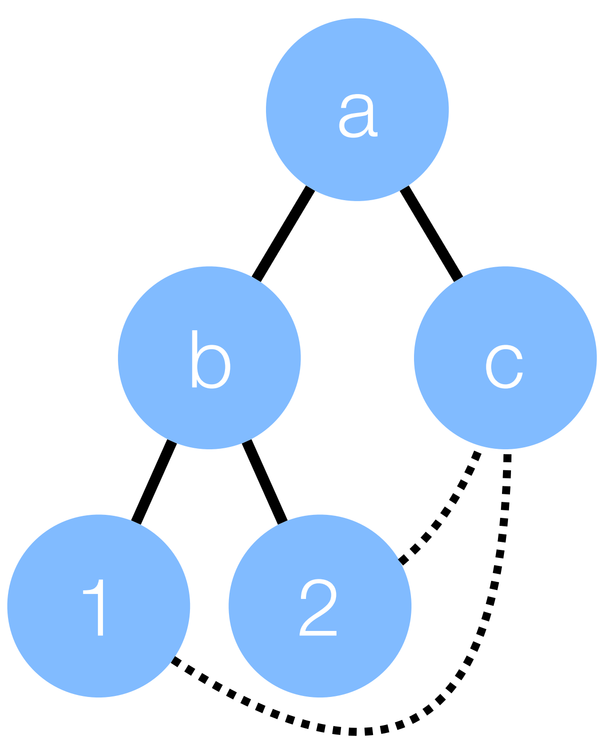
Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer



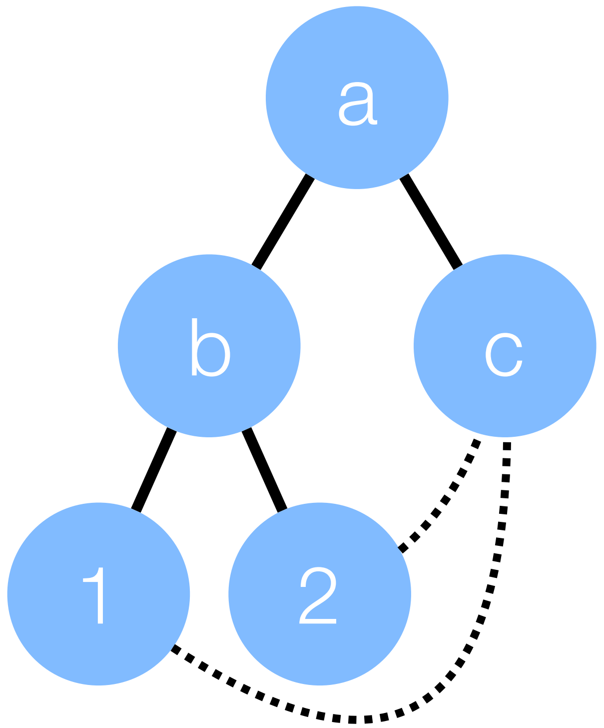
Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer
    - competitors tend to be siblings



Idea: competitors and collaborators

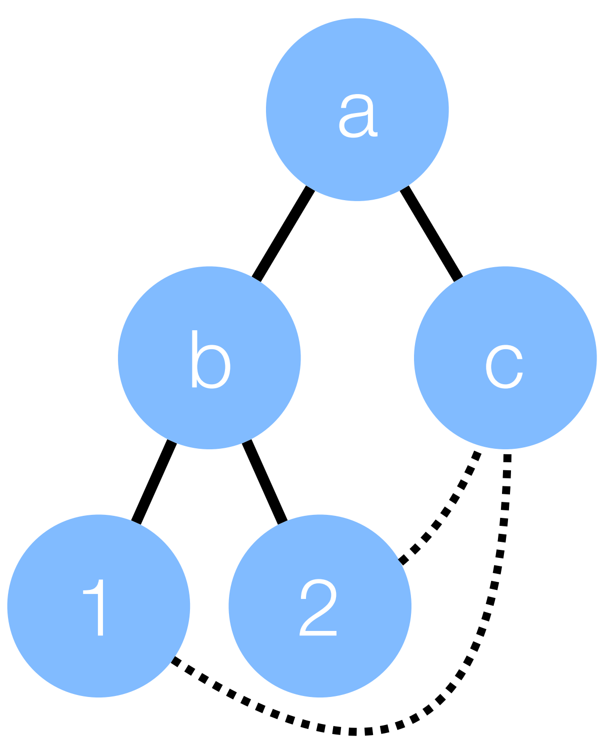
- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer
  - competitors tend to be siblings



{b, c}

Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer



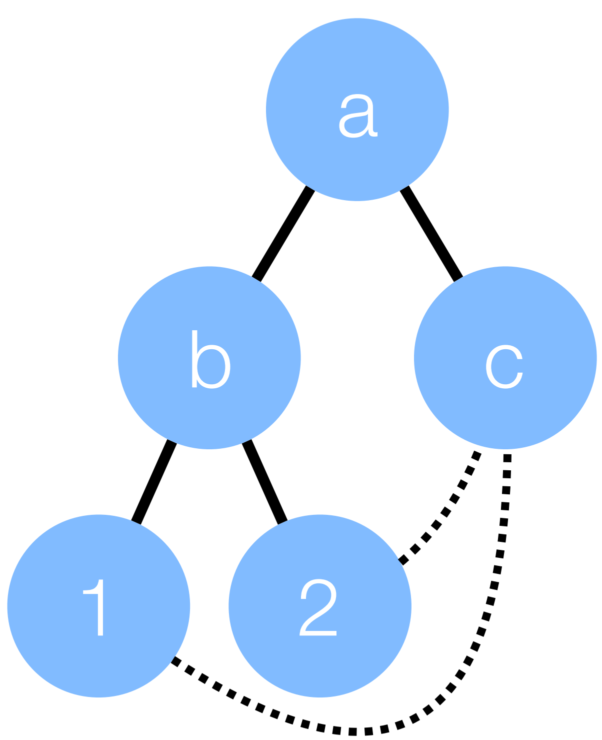
- competitors tend to be siblings

{b, c}

- collaborators tend to be in the same branch

Idea: competitors and collaborators

- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired
- Notice: a resource acquired becomes a child of the acquirer



- competitors tend to be siblings

{b, c}

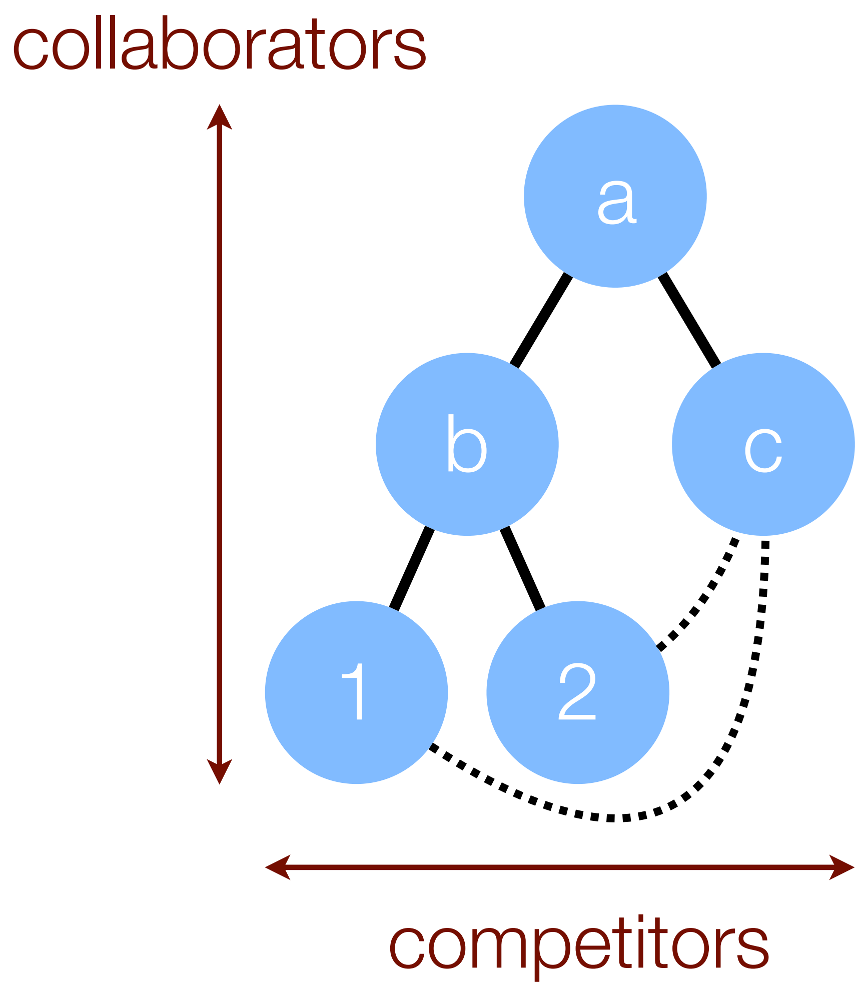
- collaborators tend to be in the same branch

{a, b, 1} {a, b, 2} {a, c}

Manifest deadlock-freedom

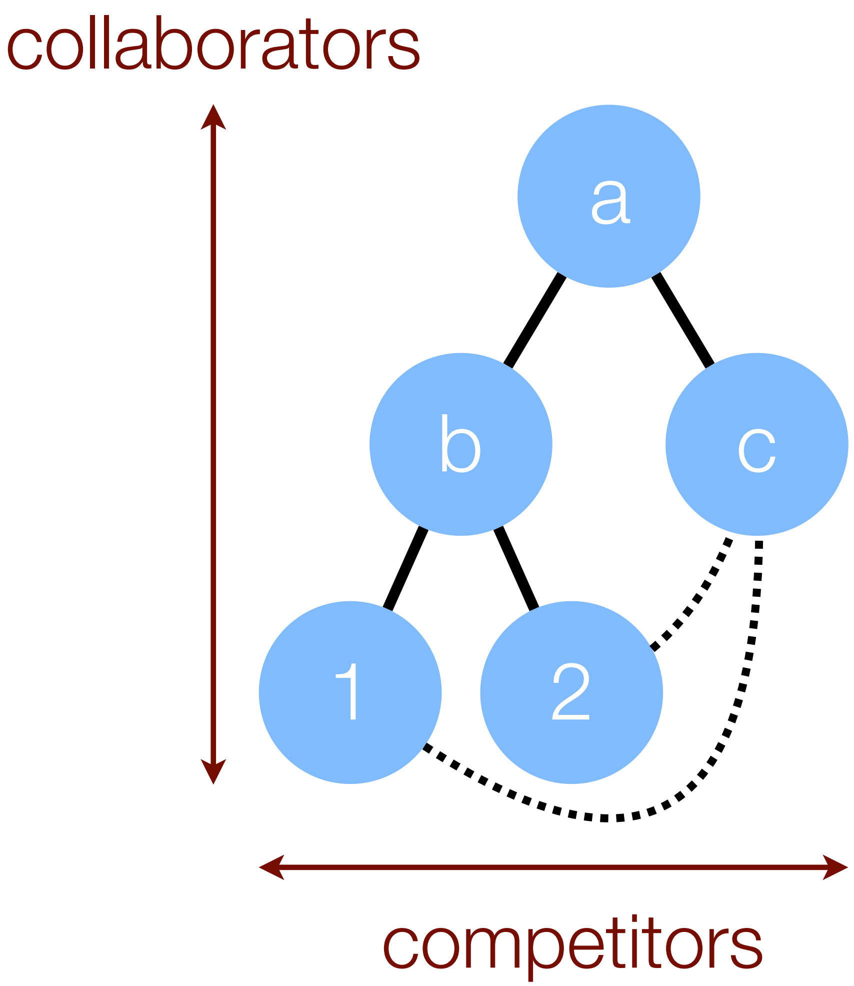
Manifest deadlock-freedom

Define type system enforcing the following invariants:



Manifest deadlock-freedom

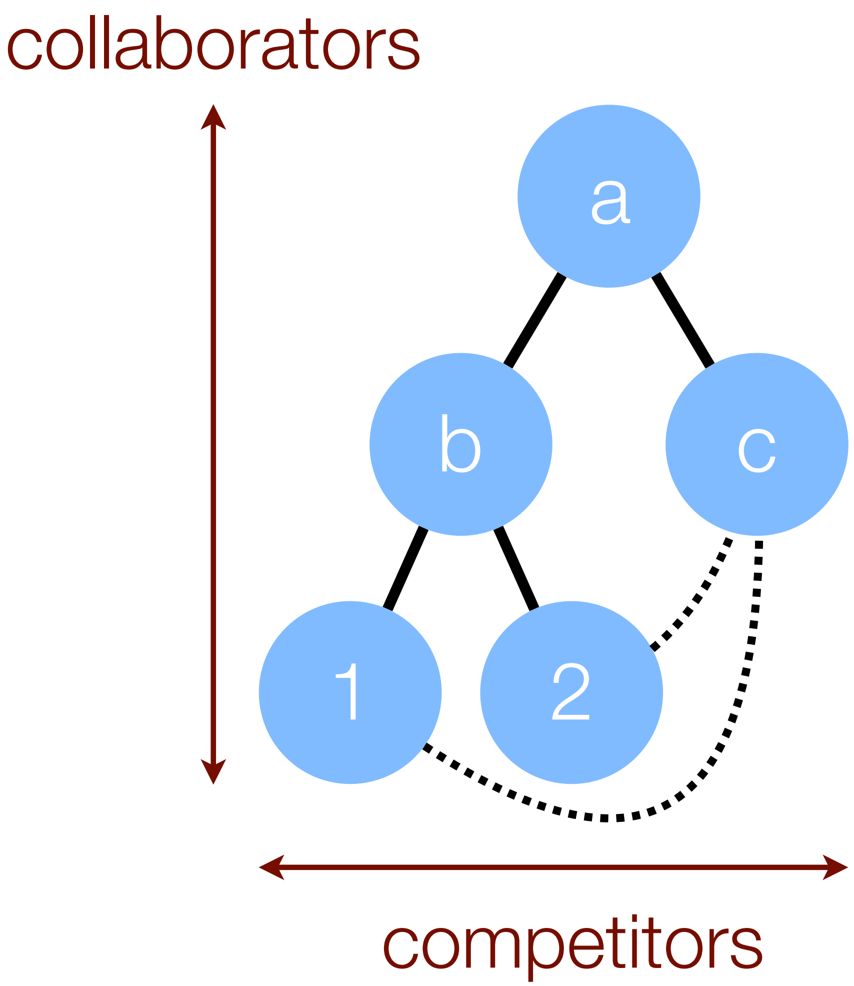
Define type system enforcing the following invariants:



A competitors employ locking-up for resources they compete for

Manifest deadlock-freedom

Define type system enforcing the following invariants:

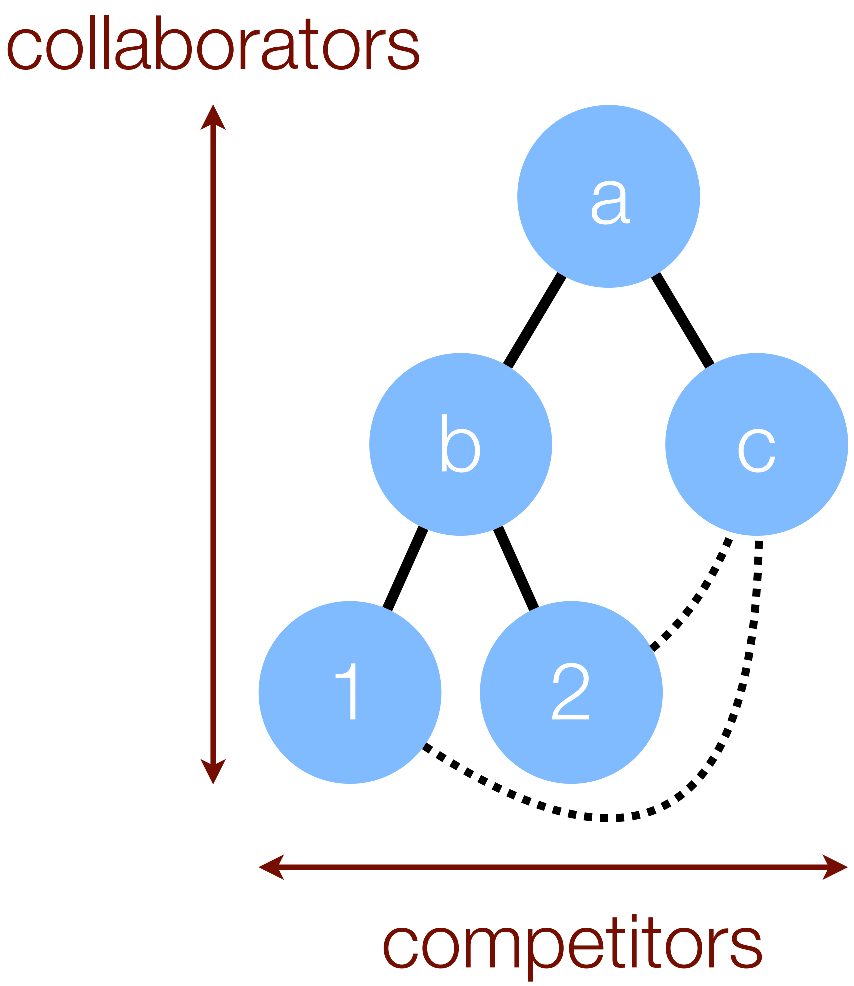


A competitors employ locking-up for resources they compete for

B collaborators acquire mutually disjoint sets of resources

Manifest deadlock-freedom

Define type system enforcing the following invariants:



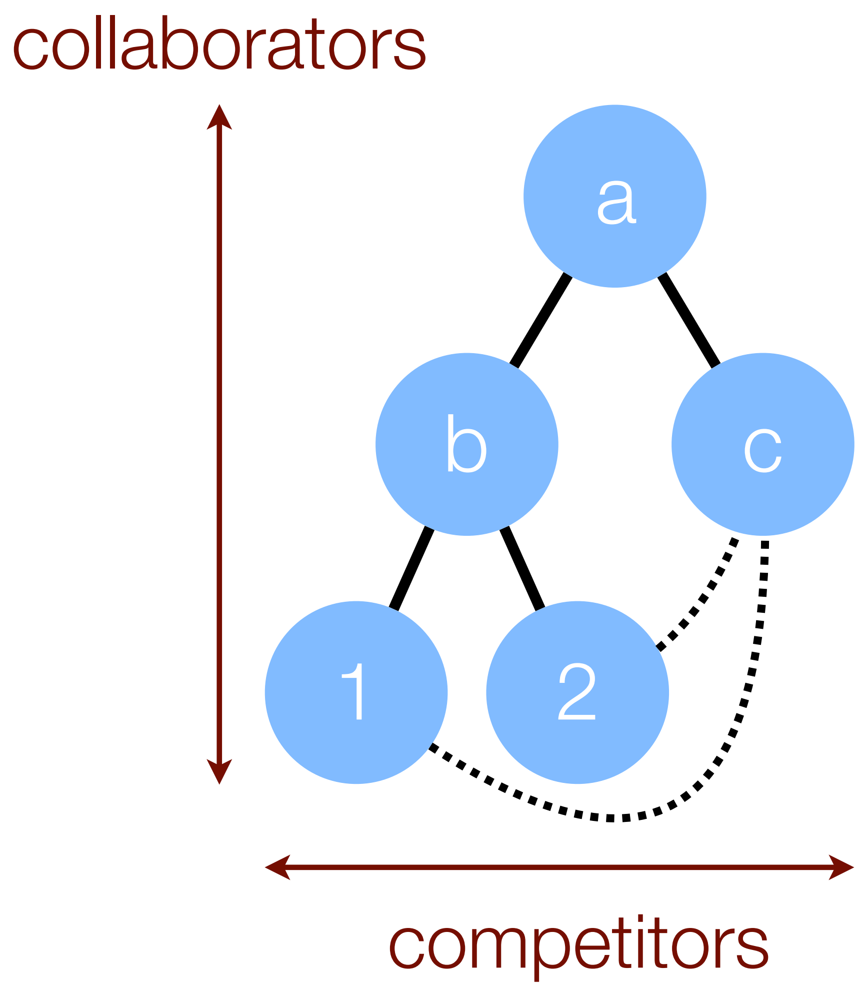
A competitors employ locking-up for resources they compete for

B collaborators acquire mutually disjoint sets of resources

C competitors have released all acquired resources when synchronizing with other competitors (“tallking-up”)

Manifest deadlock-freedom

Define type system enforcing the following invariants:



A competitors employ locking-up for resources they compete for

B collaborators acquire mutually disjoint sets of resources

C competitors have released all acquired resources when synchronizing with other competitors (“tallking-up”)

A rules out red-arrow cycles, B and C rule out red-green-arrow cycles.

Manifest deadlock-freedom

Manifest deadlock-freedom

Introduce a world, an abstract value equipped with a partial order.

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

Manifest deadlock-freedom

Introduce a world, an abstract value equipped with a partial order.

Every process invariantly resides at a world.

Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

worlds associated with process

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

Manifest deadlock-freedom

Introduce a world, an abstract value equipped with a partial order.

Every process invariantly resides at a world.

Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

self-world:
world at which process resides

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

Manifest deadlock-freedom

Introduce a world, an abstract value equipped with a partial order.

Every process invariantly resides at a world.

Every process indicates the range of worlds it may acquire.

$$\Psi;\ \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

Manifest deadlock-freedom

Introduce a world, an abstract value equipped with a partial order.

Every process invariantly resides at a world.

Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

min-world:
world of minimal resource
to be acquired

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

max-world: world
of maximal resource to be
acquired

Manifest deadlock-freedom

Introduce a world, an abstract value equipped with a partial order.

Every process invariantly resides at a world.

Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^{+} \text{ irreflexive)}$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^{+} \text{ irreflexive)}$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow_{\omega_l}^{\omega_n}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow_{\omega_l}^{\omega_n}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

world order

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^{+} \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

Manifest deadlock-freedom

- Introduce a world, an abstract value equipped with a partial order.
  - Every process invariantly resides at a world.
  - Every process indicates the range of worlds it may acquire.

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \boxed{\Phi};\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

possibly “aliased”
linear channels

Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

Express invariants A, B, and C in terms of:

Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)

Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

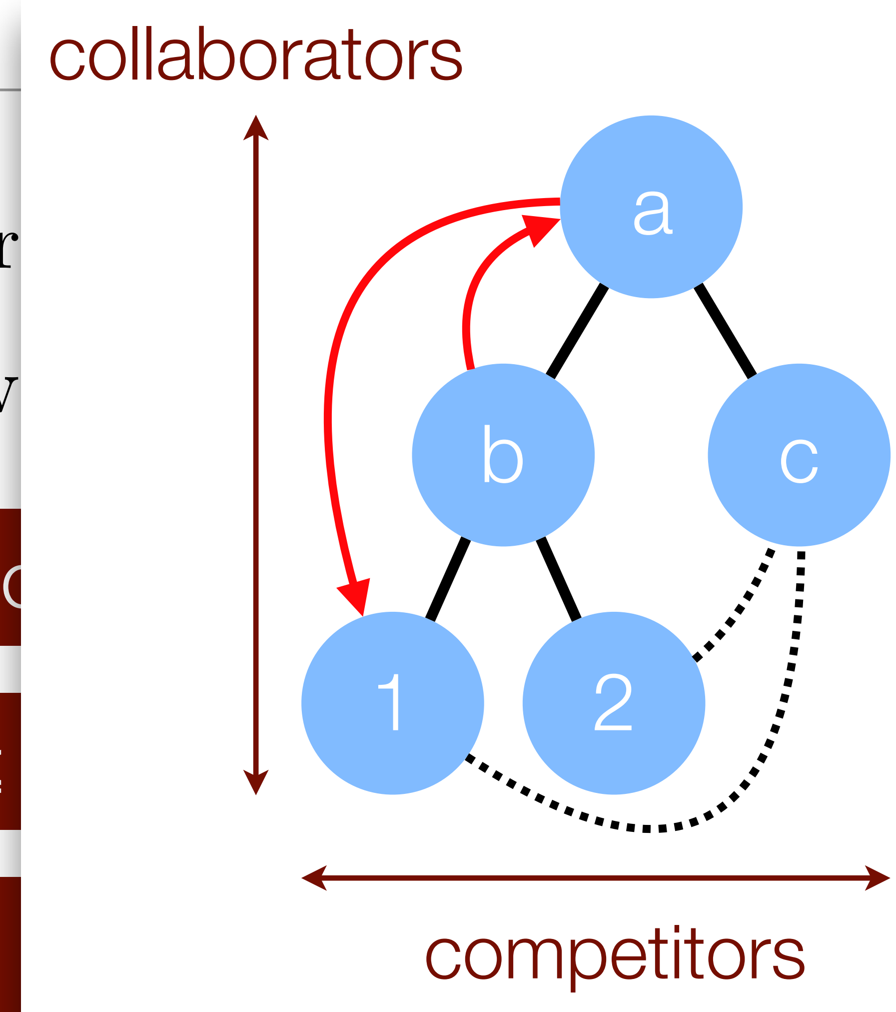
- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)

Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{wher$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{w$$

- Express invariants A, B, and C in terms c
  - min(parent) ≤ self(acquired_child) ≤
  - max(parent) < min(child)



no vertical red arrows

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

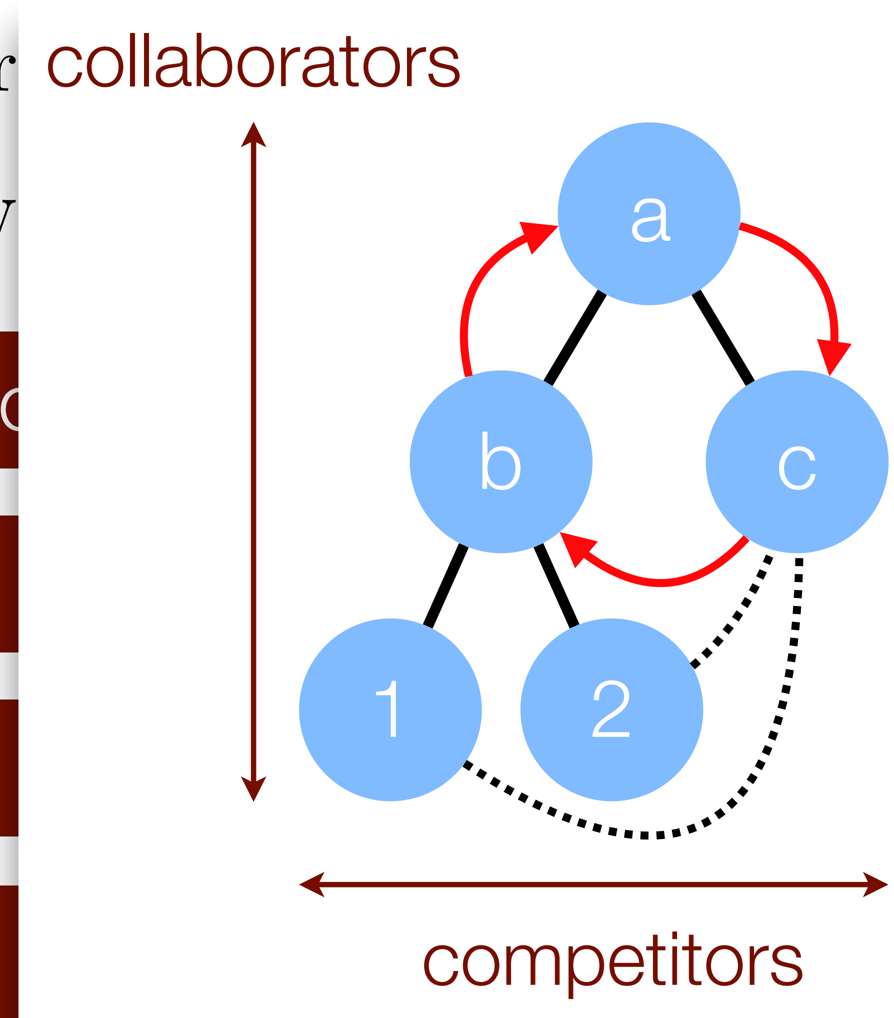
- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{wher}$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{w}$$

- Express invariants A, B, and C in terms c
  - min(parent) ≤ self(acquired_child) ≤
  - max(parent) < min(child)
  - for an acquire: lock-up



no red cycles

Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

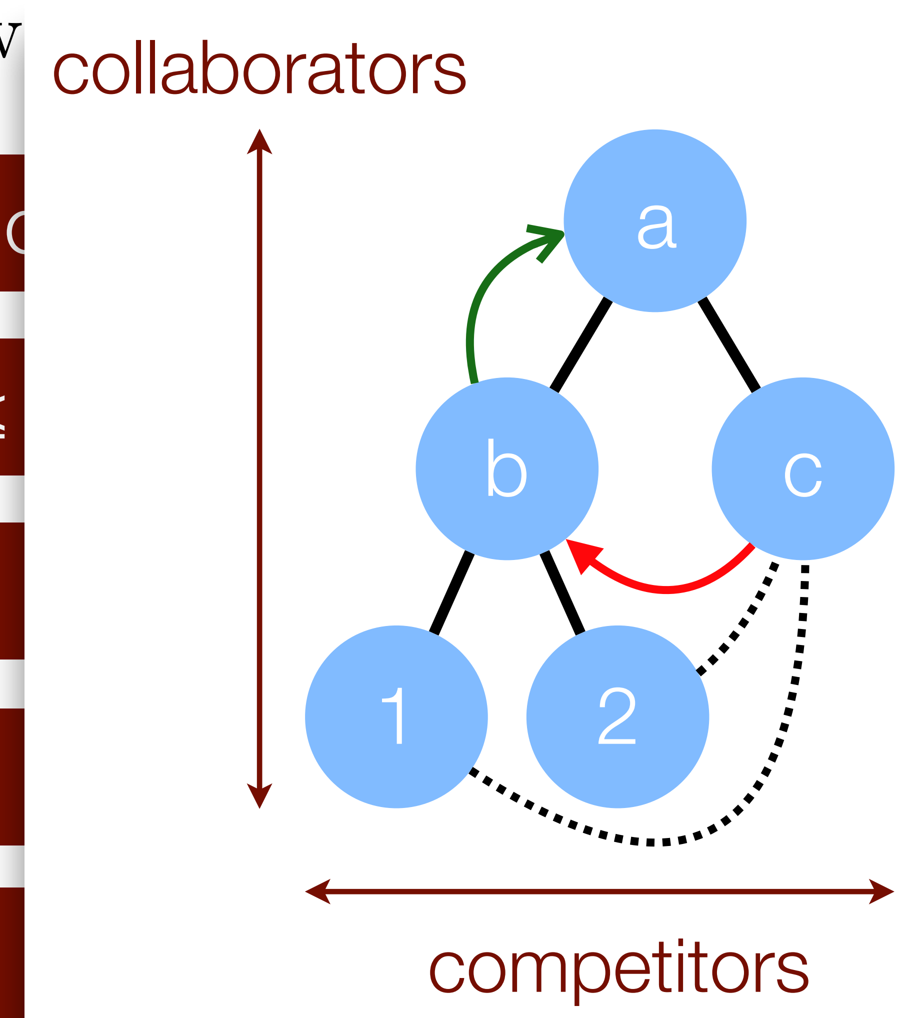
- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up
  - for right-rule: Φ must be empty

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(w}$$

- Express invariants A, B, and C in terms c
  - min(parent) ≤ self(acquired_child) ≤
  - max(parent) < min(child)
  - for an acquire: lock-up
  - for right-rule: Φ must be empty



no ingoing red and up-going green arrow

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up
  - for right-rule: Φ must be empty

Manifest deadlock-freedom

$$\Psi; \Gamma \vdash_\Sigma P :: (x_\mathsf{S} : A_\mathsf{S}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

$$\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_\mathsf{L} : A_\mathsf{L}[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)}$$

- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up
  - for right-rule: Φ must be empty
- These low-level invariants are enforced by typing.

Taking stock

Taking stock

We have a session type system that allows shared and linear channels to coexist and guarantees:

- data-race-freedom (low-level and high-level)
- protocol adherence
- deadlock-freedom

Taking stock

- We have a session type system that allows shared and linear channels to coexist and guarantees:
  - data-race-freedom (low-level and high-level)
  - protocol adherence
  - deadlock-freedom
- We have increased practicality of linear session types while maintaining their guarantees.

Current & future work

Digital contracts (with Hoffmann, Pfenning, and Das)

Digital contracts (with Hoffmann, Pfenning, and Das)

Unique application field for shared session types:

Digital contracts (with Hoffmann, Pfenning, and Das)

Unique application field for shared session types:

$$
\begin{aligned}
\mathsf{auction} = {\uparrow}_{\mathsf{L}}^{\mathsf{S}} \oplus \{&\mathsf{running} : \&\{\mathsf{bid} : \mathsf{id} \to \mathsf{money} \multimap {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}, \\
&\qquad\qquad\qquad \mathsf{cancel} : {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}\}, \\
&\mathsf{ended} : \mathsf{id} \to \oplus\{\mathsf{won} : \mathsf{lot} \otimes {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}, \\
&\qquad\qquad\qquad \mathsf{lost} : \mathsf{money} \otimes {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}\}\}
\end{aligned}
$$

Digital contracts (with Hoffmann, Pfenning, and Das)

Unique application field for shared session types:

$$
\begin{aligned}
\mathsf{auction} = {\uparrow}_{\mathsf{L}}^{\mathsf{S}} \oplus \{ &\mathsf{running} : \&\{\mathsf{bid} : \mathsf{id} \rightarrow \mathsf{money} \multimap {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}, \\
&\qquad\qquad\qquad \mathsf{cancel} : {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}\}, \\
&\mathsf{ended} : \mathsf{id} \rightarrow \oplus\{\mathsf{won} : \mathsf{lot} \otimes {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}, \\
&\qquad\qquad\qquad \mathsf{lost} : \mathsf{money} \otimes {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}\}\}
\end{aligned}
$$

Resource analysis for static prediction of execution cost.

Digital contracts (with Hoffmann, Pfenning, and Das)

Unique application field for shared session types:

$$\begin{aligned}
\mathsf{auction} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \oplus \{ &\mathsf{running} : \&\{\mathsf{bid} : \mathsf{id} \rightarrow \mathsf{money} \multimap \downarrow_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}, \\
&\qquad\qquad\qquad \mathsf{cancel} : \downarrow_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}\}, \\
&\mathsf{ended} : \mathsf{id} \rightarrow \oplus\{\mathsf{won} : \mathsf{lot} \otimes \downarrow_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}, \\
&\qquad\qquad\qquad \mathsf{lost} : \mathsf{money} \otimes \downarrow_{\mathsf{L}}^{\mathsf{S}}\mathsf{auction}\}\}
\end{aligned}$$

Resource analysis for static prediction of execution cost.

Under development: Nomos, a digital contract language based on resource-aware shared session types.

Unifying parallelism and concurrency

Unifying parallelism and concurrency

Shared session types recover expressiveness of untyped asynchronous pi-calculus [Balzer et al. CONCUR 2018]

Unifying parallelism and concurrency

Shared session types recover expressiveness of untyped asynchronous pi-calculus [Balzer et al. CONCUR 2018]

introduce nondeterminism

Unifying parallelism and concurrency

Shared session types recover expressiveness of untyped asynchronous pi-calculus [Balzer et al. CONCUR 2018]

introduce nondeterminism

linear logic session types are deterministic

Unifying parallelism and concurrency

Shared session types recover expressiveness of untyped asynchronous pi-calculus [Balzer et al. CONCUR 2018]

- introduce nondeterminism
- linear logic session types are deterministic

Opportunity for unifying framework that combines both deterministic (parallel) and nondeterministic (concurrent) computation.

Thank you for your attention!

Papers for this talk:

- Stephanie Balzer and Frank Pfenning: Manifest Sharing with Session Types. ICFP 2017.
- Stephanie Balzer, Bernardo Toninho, and Frank Pfenning: Manifest Deadlock-Freedom for Shared Session Types. ESOP 2019.
- Stephanie Balzer, Frank Pfenning, and Bernardo Toninho: A Universal Session Type for Untyped Asynchronous Communication. CONCUR 2019.