Shared Session Types for Safe, Practical Concurrency\textsuperscript{1}

Stephanie Balzer
Carnegie Mellon University

Typelevel Summit Philadelphia 2019

\textsuperscript{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

Legend: □ concurrently executing component

Shared memory

read/write

shared data
Two models for concurrent programming

Legend:
- concurrently executing component

Shared memory
- read/write
- shared data

Message-passing
- message
Two models for concurrent programming

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

- Message-passing
  - message

**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**

**Legend:** concurrently executing component

message-passing offers higher-level of abstraction
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.

![Diagram of Servo's concurrency model]

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:

- **ResourceThread**
- **ImageCache**
- **CmdReceiver**
- **Client**
- **Decoder\_1**
- **Decoder\_n**

**Legend:**
- Component, runs in separate thread
- Channel
```rust
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:
```rust
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 ()
```
```rust
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 ()

**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 ()
}
```
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 ()
```
```rust
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**
```rust
pub enum ImageCacheCommand {
    RequestImage (Url, ImageCacheChan, Option<ImageResponder>),
    GetImageIfAvailable (Url, UsePlaceholder, IpcSender<Result<Arc<Image>, ImageState>>),
    StoreDecodeImage (Url, Vec<u8>),
    ...
}
```

Legend:
- Gray component, runs in separate thread
- Channel

Image loader:

- ResourceThread
- ImageCache
- Client
- Decoder

Protocol breaches result in proliferation of `panic!` and infinite waiting.

// Clients must wait for a response before shutting down ResourceThread
Exit ()
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 ()

protocol breaches result in proliferation of panic! and infinite waiting

implicit protocol
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 ()

protocol breaches result in proliferation of panic! and infinite waiting

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

implicit protocol
```rust
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 ()

**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, 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 ()  

**protocol breaches** result in proliferation of panic! and infinite waiting

**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 define protocols of message exchange.
Session types define protocols of message exchange.

“protocol = sequence of actions”
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus \{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \{ l_i : A_i \} \]  \quad \text{external choice}

\[ \oplus \{ l_i : A_i \} \]  \quad \text{internal choice}

\[ A \rightarrow B \]  \quad \text{channel input}

\[ A \otimes B \]  \quad \text{channel output}

\[ 1 \]  \quad \text{termination}

client chooses among sending one of the labels \( l_i \)
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[
A, B \triangleq \\&\{l_i : A_i\} \quad \text{external choice}
\]
\[
\oplus\{l_i : A_i\} \quad \text{internal choice}
\]

- \(A \rightarrow B\) \quad \text{channel input}
- \(A \otimes B\) \quad \text{channel output}
- \(1\) \quad \text{termination}

Provider chooses among sending one of the labels \(l_i\).
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \{ l_i : A_i \} \quad \text{external choice} \]
\[ \oplus \{ l_i : A_i \} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus \{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]

client sends channel reference of type A
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[
A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \\
\oplus\{l_i : A_i\} \quad \text{internal choice} \\
A \rightarrow B \quad \text{channel input} \\
A \otimes B \quad \text{channel output} \\
1 \quad \text{termination}
\]

provider sends channel reference of type A
Session types define protocols of message exchange.

A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice}

\oplus\{l_i : A_i\} \quad \text{internal choice}

A \rightarrow B \quad \text{channel input}

A \otimes B \quad \text{channel output}

1 \quad \text{termination}
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[ A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]

provider terminates
Session types define protocols of message exchange.

"protocol = sequence of actions"

\[ A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[
\begin{align*}
A, B & \triangleq \&\{l_i : A_i\} & \text{external choice} \\
& \oplus\{l_i : A_i\} & \text{internal choice} \\
A \to B & & \text{channel input} \\
A \otimes B & & \text{channel output} \\
1 & & \text{termination} \\
T \to A & & \text{value input} \\
T \times A & & \text{value output} \\
T & \triangleq \text{int} \mid \text{string} \mid \ldots
\end{align*}
\]
Session types define protocols of message exchange.

“protocol = sequence of actions”

\[
A, B \quad \triangleq \quad & \{l_i : A_i\} \quad \text{external choice} \\
\oplus & \{l_i : A_i\} \quad \text{internal choice} \\
A \rightarrow B \quad \text{channel input} \\
A \otimes B \quad \text{channel output} \\
1 \quad \text{termination} \\
T \rightarrow A \quad \text{value input} \\
T \times A \quad \text{value output} \\
T \quad \triangleq \quad \text{int} \mid \text{string} \mid \ldots
\]
Session type for image loader

\[ A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \]
\[ A \oplus B \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
\[ T \rightarrow A \quad \text{value input} \]
\[ T \times A \quad \text{value output} \]

\[ T \triangleq \text{int} \mid \text{string} \mid \ldots \]
Session type for image loader

\[ A, B \triangleq \\& \{ l_i : A_i \} \quad \text{external choice} \]
\[ \oplus \{ l_i : A_i \} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
\[ T \rightarrow A \quad \text{value input} \]
\[ T \times A \quad \text{value output} \]

\[ T \triangleq \text{int} \mid \text{string} \mid \ldots \]

\[ \text{ImgCacheCmd} = \]
Session type for image loader

\[ A, B \overset{\Delta}{=} \&\{\ell_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{\ell_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
\[ T \rightarrow A \quad \text{value input} \]
\[ T \times A \quad \text{value output} \]

\[ T \overset{\Delta}{=} \text{int} \mid \text{string} \mid \ldots \]

\[ \text{ImgCacheCmd} = \& \{ \]

\[ } \]
Session type for image loader

\[ A, B \triangleq \&\{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
\[ T \rightarrow A \quad \text{value input} \]
\[ T \times A \quad \text{value output} \]

\[ T \triangleq \text{int } | \text{string } | \ldots \]

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage :} \\
\ldots \\
\ldots \\
\text{Exit :} \\
\} 
\]
Session type for image loader

\[ A, B \overset{\triangleq}{=} \&\{l_i : A_i\} \quad \text{external choice} \]
\[ \oplus\{l_i : A_i\} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
\[ T \rightarrow A \quad \text{value input} \]
\[ T \times A \quad \text{value output} \]

\[ T \overset{\triangleq}{=} \text{int} \mid \text{string} \mid \ldots \]

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \]
\[ \quad \text{Exit :} \]
\[ \} \]
Session type for image loader

\[ A, B \triangleq \& \{ l_i : A_i \} \quad \text{external choice} \]
\[ \oplus \{ l_i : A_i \} \quad \text{internal choice} \]
\[ A \rightarrow B \quad \text{channel input} \]
\[ A \otimes B \quad \text{channel output} \]
\[ 1 \quad \text{termination} \]
\[ T \rightarrow A \quad \text{value input} \]
\[ T \times A \quad \text{value output} \]

\[ T \triangleq \text{int | string | \ldots} \]

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage : string } \rightarrow \text{Requester } \rightarrow \text{ImgCacheCmd}, \ldots \]
\[ \text{Exit} : \oplus \{ \text{Running : ImgCacheCmd}, \]
\[ \text{Done : ResourceThread } \otimes \text{1} \} \]
\[ \} \]
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\quad \cdots \\
\quad \text{Exit} : \bigoplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \text{Done} : \text{ResourceThread} \otimes 1 \} \\
\} 
\]
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\quad \ldots \\
\quad \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \quad \text{Done} : \text{ResourceThread} \otimes 1 \} \\
\} 
\]
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\ldots \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \} \\
\}
\]

ImageCache:
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\ldots \\
\text{Exit} : \bigoplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \}\}
\]

ImageCache: ImgCacheCmd
Session type for image loader

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \]

\[ \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \]

\[ \text{Done} : \text{ResourceThread} \otimes 1 \} \]

\]

ImageCache: ImgCacheCmd
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage : string } \rightarrow \text{Requester } \rightarrow \text{ImgCacheCmd}, \\
\ldots \\
\text{Exit} : \oplus \{ \text{Running : ImgCacheCmd}, \\
\text{Done : ResourceThread} \otimes 1 \} \\
\}
\]

ImageCache:
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \to \text{Requester} \to \text{ImgCacheCmd},
\]

\[
\ldots
\]

\[
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd},
\]

\[
\text{Done} : \text{ResourceThread} \otimes 1 \}
\]

\[
\}
\]

\[
\text{ImageCache}: \text{string} \to \text{Requester} \to \text{ImgCacheCmd}
\]
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \circ \text{ImgCacheCmd}, \\
\ldots \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \text{Done} : \text{ResourceThread} \otimes 1 \} \\
\}
\]

\[
\text{ImageCache}: \text{string} \rightarrow \text{Requester} \rightarrow \circ \text{ImgCacheCmd}
\]
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\ DOI \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\ DOI \\
\text{Done} : \text{ResourceThread} \odot 1 \}
\}
\]

ImageCache: Requester \rightarrow ImgCacheCmd
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\ldots \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \} \\
\}
\]

\text{ImageCache}: \text{Requester} \rightarrow \text{ImgCacheCmd}
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \to \text{Requester} \to \text{ImgCacheCmd}, \\
\cdots \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \} \\
\}
\]

ImageCache: ImgCacheCmd
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage : string } \to \text{Requester } \to \text{ImgCacheCmd}, \\
\ldots \text{Exit } : \oplus \{ \text{Running } : \text{ImgCacheCmd}, \\
\text{Done } : \text{ResourceThread } \otimes 1 \} \}
\]

**ImageCache:** ImgCacheCmd
Session type for image loader

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \} \]

\[ \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \]
\[ \text{Done} : \text{ResourceThread} \otimes 1 \} \]

\[ \text{ImageCache: } \oplus \{ \text{Running} : \text{ImgCacheCmd}, \text{Done} : \text{ResourceThread} \otimes 1 \} \]
Session type for image loader

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage : string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \ \\
\quad \quad \text{Exit} : \oplus \{ \text{Running : ImgCacheCmd}, \text{Done : ResourceThread} \otimes 1 \} \} \]

ImageCache: \( \oplus \{ \text{Running : ImgCacheCmd, Done : ResourceThread} \otimes 1 \} \)
Session type for image loader

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \text{Done} : \text{ResourceThread} \otimes 1 \} \}
\]

\text{ImageCache: ImgCacheCmd}
Session type for image loader

ImageCache: ImgCacheCmd

- ResourceThread
- CmdReceiver
- ImageCache
- Client
- Decoder_1
- \ldots
- Decoder_n
Session type for image loader

components change their session type along with message exchange

ImageCache: 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

Provide strong guarantees:

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

exactly one client
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

\[ \text{ResourceThread} \rightarrow \text{ImageCache} \rightarrow \text{CmdReceiver} \rightarrow \text{Decoder}_1 \rightarrow \ldots \rightarrow \text{Decoder}_n \rightarrow \text{Client} \]

both CmdReceiver and Client communicate with ImageCache
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

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\ldots \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \} \\
\}
\]
Challenges of sharing

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage : string} \to \text{Requester} \to \text{ImgCacheCmd}, \ldots \text{Exit : } \oplus \{ \text{Running : ImgCacheCmd}, \text{Done : ResourceThread } \otimes 1 \} \} \]

ImageCache: ImgCacheCmd

![Diagram showing the relationships betweenImgCacheCmd, RequestImage, Requester, ImgCacheCmd, Exit, Running, ResourceThread, and Decoder. The diagram illustrates how ImgCacheCmd is processed through various threads and decoders.]
Challenges of sharing

ImgCacheCmd = \& \{ RequestImage : string \rightarrow Requester \rightarrow ImgCacheCmd, \\
\ldots \\
Exit : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \text{Done} : \text{ResourceThread} \otimes 1 \} \}

ImageCache: ImgCacheCmd
Challenges of sharing

\[ \text{ImgCacheCmd} \equiv \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \\]
\[ \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \]
\[ \text{Done} : \text{ResourceThread} \otimes 1 \} \]

\[ \text{ImageCache} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd} \]
Challenges of sharing

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \to \text{Requester} \to \circ \text{ImgCacheCmd},
\]

\[
\ldots
\]

\[
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd},
\]

\[
\text{Done} : \text{ResourceThread} \otimes 1 \}
\]

\[
\}
\]

ImageCache: \text{string} \to \text{Requester} \to \circ \text{ImgCacheCmd}

![Diagram of network components including ResourceThread, Exit, CmdReceiver, Client, Decoder_1, ..., Decoder_n, and protocol violating label.]

protocol violated!
Challenges of sharing

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \ldots \]

\[ \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \]

\[ \text{Done} : \text{ResourceThread} \otimes 1 \} \]

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.

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.

Legend: 
- dotted line: shared channel
- solid line: 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.

Legend:

- dotted line: shared channel
- solid line: 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.

Legend: ............................ shared channel  —— linear channel
Have we restored protocol adherence?
Have we restored protocol adherence?

\[ \text{ImgCacheCmd} = \&\{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \]
\[ \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \]
\[ \text{Done} : \text{ResourceThread} \otimes 1 \} \} \]

Legend: ・・・・ shared channel  ——— linear channel
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \&\{ \text{RequestImage} : \text{string} \to \text{Requester} \to \text{ImgCacheCmd}, \\
\quad \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \quad \text{Done} : \text{ResourceThread} \otimes 1 \} \}
\]

**Legend:** .......................... shared channel  —— linear channel
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \&\{ \text{RequestImage : string } \rightarrow \text{Requester } \mapsto \text{ImgCacheCmd}, \\
\text{Exit} : \oplus \{ \text{Running : ImgCacheCmd}, \\
\text{Done : ResourceThread } \otimes 1 \}\}
\]

ImageCache: ImgCacheCmd

Legend:  
- dashed line: shared channel  
- solid line: linear channel
Have we restored protocol adherence?

\[ \text{ImgCacheCmd} = \& \{ \text{RequestImage : string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \]
\[ \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \]
\[ \text{Done} : \text{ResourceThread} \otimes 1 \} \}

ImageCache: ImgCacheCmd

Legend: \(\cdots\cdots\) shared channel  \(\longrightarrow\) linear channel
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\quad \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \quad \text{Done} : \text{ResourceThread} \otimes 1 \} \}
\]

**ImageCache**: \( \text{ImgCacheCmd} \)

---

**Legend:**
- Dashed line: shared channel
- Solid line: linear channel
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage : string} \rightarrow \text{Requester} \rightarrow \circ \text{ImgCacheCmd}, \\
\text{Exit} : \bigoplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \} \}
\]

\text{ImageCache: string} \rightarrow \text{Requester} \rightarrow \circ \text{ImgCacheCmd}

\text{Legend: } \cdots \text{ shared channel} \quad \longrightarrow \text{ linear channel}
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \&\{\text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\text{Exit} : \oplus \{\text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes \text{1}\}\}
\]

**ImageCache**: string → Requester → ImgCacheCmd

---

**Legend**: 
- ----- shared channel
- --- linear channel
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \&\{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\text{Done} : \text{ResourceThread} \otimes 1 \}\}
\]

ImageCache: string → Requester → ImgCacheCmd

**Legend:**
- ······ shared channel
- ——— linear channel
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \& \{\text{RequestImage : string } \rightarrow \text{Requester } \rightarrow \text{ImgCacheCmd}, \\
\text{Exit} : \oplus \{\text{Running : ImgCacheCmd}, \\
\text{Done : ResourceThread } \otimes 1\}\}
\]

ImageCache: string \rightarrow \text{Requester } \rightarrow \text{ImgCacheCmd}

Legend: \quad \cdots \quad \text{shared channel} \quad \longrightarrow \quad \text{linear channel}
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage : string \rightarrow Requester \rightarrow \circ \text{ImgCacheCmd}}, \\
\text{Exit : } \oplus \{ \text{Running : ImgCacheCmd}, \\
\text{Done : ResourceThread } \otimes 1 \} \}\]

ImageCache: string \rightarrow Requester \rightarrow \circ \text{ImgCacheCmd}

Legend:  
\[\text{------ shared channel} \quad \text{--- linear channel}\]
Have we restored protocol adherence?

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\quad \text{Exit} : \oplus \{ \text{Running} : \text{ImgCacheCmd}, \\
\quad \quad \text{Done} : \text{ResourceThread} \otimes 1 \} \}
\]

**ImageCache**: string → Requester → 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.

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\quad \text{Exit} : \oplus \{ \text{Done} : \text{ResourceThread} \otimes 1, \\
\quad \quad \text{Running} : \text{ImgCacheCmd} \}\}
\]
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.

\[
\text{acquire}
\]

\[
\text{ImgCacheCmd} = \&\{\text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd}, \\
\text{Exit} : \oplus \{\text{Done} : \text{ResourceThread} \odot 1, \\
\text{Running} : \text{ImgCacheCmd}\}\}
\]
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.

\[
\begin{align*}
\text{ImgCacheCmd} &= \&\{\text{RequestImage : string} \to \text{Requester} \to \text{ImgCacheCmd}, \\
& \quad \text{Exit : } \oplus \{\text{Done : ResourceThread } \otimes 1, \\
& \quad \quad \quad \text{Running : ImgCacheCmd}\}\}
\end{align*}
\]
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.

\[
\text{ImgCacheCmd} = \&\{ \text{RequestImage : string} \rightarrow \text{Requester} \rightarrow \circ \text{ImgCacheCmd}, \\
\quad \text{Exit} : \oplus \{ \text{Done : ResourceThread} \otimes 1, \\
\quad \text{Running : ImgCacheCmd} \}\}
\]
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.

\[ \text{ImgCacheCmd} = \&\{\text{RequestImage: string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd},\]
\[ \text{Exit: } \oplus \{\text{Done: ResourceThread } \otimes 1,\]
\[ \text{Running: ImgCacheCmd}\}\}

acquire

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.

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage : string \rightarrow Requester \rightarrow ImgCacheCmd,} \\
\quad \text{Exit : } \oplus \{ \text{Done : ResourceThread } \otimes 1,} \\
\quad \text{Running : ImgCacheCmd} \} \}
\]

acquire

release ✓

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.

```
ImgCacheCmd = \&\{RequestImage : string \rightarrow Requester \leftarrow ImgCacheCmd, Exit : \oplus \{Done : ResourceThread \otimes 1, Running : ImgCacheCmd\}\}
```

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.

\[
\text{ImgCacheCmd} = \&\{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \circ \text{ImgCacheCmd}, \\
\text{Exit} : \oplus \{ \text{Done} : \text{ResourceThread} \otimes 1, \\
\text{Running} : \text{ImgCacheCmd} \} \}
\]
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

\[
\text{ImgCacheCmd} = \&\{{\text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{ImgCacheCmd},}
\]
\[
\text{Exit} : \oplus \{\text{Done} : \text{ResourceThread} \otimes 1, \linebreak \text{Running} : \text{ImgCacheCmd}\}\}
\]
Manifest sharing

\[ \text{ImgCacheCmd} = \{ \text{RequestImage} : \text{string} \Rightarrow \text{Requester} \mapsto \text{ImgCacheCmd}, \text{Exit} : \bigoplus \{ \text{Done} : \text{ResourceThread} \otimes 1, \text{Running} : \text{ImgCacheCmd} \} \} \]

Legend:  
- purple: shared phase  
- blue: linear phase
Manifest sharing
Manifest sharing

Stratify session types into a linear and shared layer
Stratify session types into a linear and shared layer

\[ A_S \triangleq \]
\[ A_L, B_L \triangleq \bigoplus \{ l : A_L \} \ | \ A_L \otimes B_L \ | \ 1 \ | \]
\[ \& \{ l : A_L \} \ | \ A_L \leadsto B_L \]
Manifest sharing

Stratify session types into a linear and shared layer

Connect layers with modalities going back and forth

\[
A_S \triangleq \quad
\]

\[
A_L, B_L \triangleq \oplus \{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid
\]

\[
& \& \{l : A_L\} \mid A_L \dashv B_L
\]
Manifest sharing

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

\[
A_S \triangleq \uparrow^S_L A_L
\]

\[
A_L, B_L \triangleq \bigoplus \{l : A_L\} \mid A_L \otimes B_L \mid 1 \mid \\
& \& \{l : A_L\} \mid A_L \to B_L \mid \downarrow^S_L A_S
\]
Manifest sharing

Stratify session types into a linear and shared layer

Connect layers with modalities going back and forth

Support communication of shared channels

\[
A_S \triangleq \uparrow^S_l A_L
\]

\[
A_L, B_L \triangleq \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \\
\& \{ l : A_L \} \mid A_L \twoheadrightarrow B_L \mid \downarrow^S_l A_S
\]
Manifest sharing

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

\[
A_S \triangleq \uparrow^S_L A_L
\]

\[
A_L, B_L \triangleq \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L
\]

\[
\& \{ l : A_L \} \mid A_L \to B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S. B_L
\]
Manifest sharing

$A_S \triangleq \uparrow^S_{\downarrow L} A_L$

$A_L, B_L \triangleq \oplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \mid$

$\& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S_{\uparrow L} A_S \mid \Pi x : A_S. B_L$
Manifest sharing

\[
A_S \triangleq \leftarrow^S A_L
\]

\[
A_L, B_L \triangleq \oplus \{ l: A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x: A_S. B_L \\
\& \{ l: A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S A_S \mid \Pi x: A_S. B_L
\]

\[
\text{ImgCacheCmd} = \& \{ \text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \text{Exit} : \oplus \{ \text{Done} : \text{ResourceThread} \otimes 1, \text{Running} : \text{ImgCacheCmd} \} \}
\]
Manifest sharing

\[
A_S \triangleq \uparrow^S A_L
\]

\[
A_L, B_L \triangleq \oplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \mid \\
\& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S A_S \mid \Pi x : A_S. B_L
\]

\[
\text{ImgCacheCmd} = \uparrow^S \& \{ \text{RequestImage : string } \rightarrow \text{Requester } \rightarrow \}
\]

\[
\text{Exit : } \oplus \{ \text{Done : ResourceThread } \otimes 1, \text{Running : ImgCacheCmd} \}
\]
Manifest sharing

\[
A_S \triangleq \uparrow^S \downarrow^L A_L
\]

\[
A_L, B_L \triangleq \bigoplus \{l : \overline{A_L}\} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S \cdot B_L \mid
\]

\[
\& \{l : A_L\} \mid A_L \leftarrow B_L \mid \downarrow^S A_S \mid \Pi x : A_S \cdot B_L
\]

\[
\text{ImgCacheCmd} = \uparrow^S \& \{\text{RequestImage} : \text{string} \rightarrow \text{Requester} \rightarrow \downarrow^S \text{ImgCacheCmd}, \text{Exit} : \bigoplus \{\text{Done} : \text{ResourceThread} \otimes 1, \text{Running} : \downarrow^S \text{ImgCacheCmd}\}\}
\]
Manifest sharing

\[
A_S \triangleq \uparrow^S_L A_L
\]

\[
A_L, B_L \triangleq \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S \cdot B_L \mid
\]

\[
\& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S \cdot B_L
\]

\[
\text{ImgCacheCmd} = \uparrow^S_L \& \{ \text{RequestImage : string} \rightarrow \text{Requester} \rightarrow \downarrow^S_L \text{ImgCacheCmd}, \text{Exit} : \bigoplus \{ \text{Done : ResourceThread} \otimes 1, \text{Running} : \downarrow^S_L \text{ImgCacheCmd} \}\}
\]

Up and down shifts denote acquire and release, resp.
Typing judgments

\[ A_S \triangleq \uparrow^S L A_L \]

\[ A_L, B_L \triangleq \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S . B_L \mid \]

\[ \& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S L A_S \mid \Pi x : A_S . B_L \]
Typing judgments

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

\[
A_S \doteq \uparrow^S \downarrow^L A_L
\]

\[
A_L, B_L \doteq \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \mid
\& \{ l : A_L \} \mid A_L \multimap B_L \mid \downarrow^S A_S \mid \Pi x : A_S. B_L
\]

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

\[
\Gamma \vdash \Sigma P :: (x_S : A_S)
\]

\[
\Gamma; \Delta \vdash \Sigma P :: (x_L : A_L)
\]
Based on correspondence between intuitionistic linear logic and session-typed pi-calculus:

\[
\begin{align*}
A_S \triangleq & \uparrow^S_A L \\
A_L, B_L \triangleq & \bigoplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \\
& \& \& \bigland \{ l : A_L \} \mid A_L \multimap B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S. B_L
\end{align*}
\]

"Process \( P \) provides session of type \( A_m \) along \( x_m \) using channels in (\( \Gamma \) and) \( \Delta \)."
Typing judgments

\[
\begin{align*}
A_S & \triangleq \uparrow^S_L A_L \\
A_L, B_L & \triangleq \bigoplus \{l : A_L\} \mid A_L \otimes B_L \mid \mathbf{1} \mid \exists x : A_S. B_L \mid \\
& \quad \& \{l : A_L\} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S. B_L
\end{align*}
\]

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

```
Γ ⊩ P :: (x_S : A_S)
Γ; Δ ⊩ P :: (x_L : A_L)
```

“Process \( P \) provides session of type \( A_m \) along \( x_m \) using channels in (\( Γ \) and) \( Δ \).”
Typing judgments

\[
\begin{align*}
A_S & \triangleq \uparrow^S_L A_L \\
A_L, B_L & \triangleq \oplus \{ l : A_L \} \mid A_L \otimes B_L \mid 1 \mid \exists x : A_S. B_L \\
& \hspace{1cm} \& \{ l : A_L \} \mid A_L \rightarrow B_L \mid \downarrow^S_L A_S \mid \Pi x : A_S. B_L
\end{align*}
\]

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

\[
\Gamma \vdash \Sigma \ P :: (x_S : A_S) \\
\Gamma ; \ \Delta \vdash \Sigma \ P :: (x_L : A_L)
\]

“Process \( P \) provides session of type \( A_m \) along \( x_m \) using channels in (\( \Gamma \) and) \( \Delta \).”
Typing judgments

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

**linear (substructural) context**

\[\Gamma \vdash \Sigma \ P :: (x_S : A_S)\]

\[\Gamma; \Delta \vdash \Sigma \ P :: (x_L : A_L)\]

“Process \(P\) provides session of type \(A_m\) along \(x_m\) using channels in \((\Gamma \text{ and} ) \Delta\).”
Typing of acquire
Typing of acquire

\[
\Gamma, x_s : \uparrow^S A_L; \ \Delta, x_l : A_L \vdash \Sigma \ Q x_l :: (z_L : C_L) \\
\Gamma, x_s : \uparrow^S A_L; \ \Delta \vdash \Sigma \ x_l \leftarrow \text{acquire} \ x_s ; Q x_l :: (z_L : C_L)
\]
Typing of acquire

\[
\Gamma, x_S : \uparrow^S L A_L; \quad \Delta, x_L : A_L \vdash \Sigma Q x_L :: (z_L : C_L) \\
\Gamma, x_S : \uparrow^S L A_L; \quad \Delta \vdash \Sigma x_L \leftarrow \text{acquire } x_S ; Q x_L :: (z_L : C_L) \\
\text{(T-} \uparrow^S L L)\quad (T-\uparrow^S L L)
\]
Typing of acquire

\[
\Gamma, x_S : \uparrow^S L A_L; \quad \Delta, x_L : A_L \vdash_{\Sigma} Q_{x_L} :: (z_L : C_L)
\]

\[
\Gamma, x_S : \uparrow^S L A_L; \quad \Delta \vdash_{\Sigma} x_L \leftarrow \text{acquire } x_S ; Q_{x_L} :: (z_L : C_L)
\]

\[
\Gamma; \cdot \vdash_{\Sigma} P_{x_L} :: (x_L : A_L)
\]

\[
\Gamma \vdash_{\Sigma} x_L \leftarrow \text{accept } x_S ; P_{x_L} :: (x_S : \uparrow^S L A_L)
\]
Typing of acquire

\[
\begin{align*}
\Gamma, x_S : \mathbin{\uparrow^S_L} A_L; \quad & \Delta, x_L : A_L \vdash Q x_L :: (z_L : C_l) \\
\Gamma, x_s : \mathbin{\uparrow^S_L} A_L; \quad & \Delta \vdash x_L \leftarrow \text{acquire} \ x_S; \ Q x_L :: (z_L : C_l)
\end{align*}
\]

\[
(T-\mathbin{\uparrow^S_{LL}})
\]

\[
\begin{align*}
\Gamma; & \vdash P x_L :: (x_L : A_L) \\
\Gamma & \vdash x_L \leftarrow \text{accept} \ x_S; \ P x_L :: (x_S : \mathbin{\uparrow^S_L} A_L)
\end{align*}
\]

\[
(T-\mathbin{\uparrow^S_{LR}})
\]
Typing of release
Typing of release

\[
\Gamma, x_S : A_S; \Delta \vdash \Sigma Q x_S :: (z_L : C_L)
\]

\[
\Gamma; \Delta, x_L : \downarrow_L^s A_S \vdash \Sigma x_S \leftarrow \text{release } x_L ; Q x_S :: (z_L : C_L)
\]
Typing of release

\[
\frac{\Gamma, x_S : A_S; \quad \Delta \vdash \sum Q x_S :: (z_L : C_L)}{\Gamma; \quad \Delta, x_L : \downarrow L_A S \vdash \sum x_S \leftarrow \text{release } x_L ; Q x_S :: (z_L : C_L)}
\]
Typing of release

\[
\frac{\Gamma, x_S : A_S; \Delta \vdash \Sigma Q_x : (z_L : C_L)}{\Gamma; \Delta, x_L \downarrow L A_S \vdash \Sigma x_S \leftarrow \text{release } x_L ; Q_x : (z_L : C_L)} \quad (T_{\downarrow L}^s)
\]

\[
\frac{\Gamma \vdash \Sigma P_x : (x_S : A_S)}{\Gamma; \cdot \vdash \Sigma x_S \leftarrow \text{detach } x_L ; P_x : (x_L : \downarrow L^s A_S)} \quad (T_{\downarrow L}^s)
\]

58
Typing of release

\[
\frac{\Gamma, x_S : A_S; \Delta \vdash \Sigma Q_{x_S} :: (z_L : C_L)}{\Gamma; \Delta, x_L : \downarrow^s L A_S \vdash \sum x_S \leftarrow \text{release } x_L ; Q_{x_S} :: (z_L : C_L)} (T-\downarrow^s L L)
\]

\[
\frac{\Gamma \vdash \Sigma P_{x_S} :: (x_S : A_S)}{\Gamma; \cdot \vdash \sum x_S \leftarrow \text{detach } x_L ; P_{x_S} :: (x_L : \downarrow^s L A_S)} (T-\downarrow^s L R)
\]
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
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.

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
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 → b  "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 → b “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: linear channel  
a ➔ b  “a waits for b”
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**
- **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
- **○** 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
- 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”
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

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

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

Legend:  
- Linear channel: black solid line
- Linear process: blue circle
- Shared process: pink circle
- Shared channel: dotted line

"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 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 channel
- Shared process
- “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 channel
- shared process

\[ a \text{ waits for } b \text{ to release resource} \]
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 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: \( a \rightarrow b \) “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:  \( a \rightarrow b \)  “\( a \) 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: \( a \rightarrow b \) “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: \( a \rightarrow b \) “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: \( a \rightarrow b \) “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: \( a \rightarrow b \) “\( a \) waits for \( b \) to synchronize”
- waiting to release: \( a \rightarrow b \) “\( a \) 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:

collaborators

competitors
Manifest deadlock-freedom

Define type system enforcing the following invariants:

collaborators

competitors employ locking-up for resources they compete for
Define type system enforcing the following invariants:

- Competitors employ locking-up for resources they compete for:
  - Competitors (A)

- Collaborators acquire mutually disjoint sets of resources:
  - Collaborators (B)
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 ("talking-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 (“talking-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.

\[
\Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \downarrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]
Manifest deadlock-freedom

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

Every process invariants resides at a world.

Every process indicates the range of worlds it may acquire.

\[
\Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_l : A_l[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]
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.

$$\Psi; \Gamma \vdash \Sigma P :: (x_s : A_s[\omega_k \uparrow \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

$$\Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \uparrow \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive})$$

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.

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_S : A_s[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_l[\omega_k \updownarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]
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.

\[ \Psi; \Gamma \vdash_\Sigma P :: (x_s : A_s[\omega_k \sqsupset \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_L : A_L[\omega_k \sqsupset \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]
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.

\[ \Psi; \Gamma \vdash \Sigma P :: (x_s : A_s[\omega_k \downarrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \downarrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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.

\[
\Psi; \Gamma \vdash_{\Sigma} P :: (x_S : A_S[\omega_k \uparrow \omega_l]) \quad \text{(where } \Psi^+ \text{ irreflexive})
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \quad \text{(where } \Psi^+ \text{ irreflexive})
\]
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.

\[ \Psi; \Gamma \vdash \Sigma P :: (x_S : A_S[\omega_k \downarrow\omega_n]) \text{ (where } \Psi^+ \text{ irreflexive)} \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \downarrow\omega_n]) \text{ (where } \Psi^+ \text{ irreflexive)} \]
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.

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \uparrow^\omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow^\omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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.

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \downarrow_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_l : A_l[\omega_k \uparrow_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]
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.

\[ \Psi; \Gamma \vdash \Sigma P :: (x_S : A_S[\omega_k \uparrow^\omega_n \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \downarrow^\omega_n \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]
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.

\[
\Psi; \Gamma \vdash \Sigma P :: (x_s : A_s[\omega_k \uparrow_{\omega_l}]) \text{ (where } \Psi^+ \text{ irreflexive)}
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \uparrow_{\omega_L}]) \text{ (where } \Psi^+ \text{ irreflexive)}
\]

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.

\[ \Psi ; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \uparrow \omega^n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi ; \Gamma ; \Phi ; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \downarrow \omega^n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]
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.

\[
\Psi; \Gamma \vdash \Sigma P :: (x_S : A_S[\omega_k \downarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \uparrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]
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.

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_S : A_S[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]
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.

\[
\Psi; \Gamma \vdash_\Sigma P :: (x_S : A_S[\omega_k \downarrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive})
\]
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.

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \uparrow_{\omega_l} \omega_n]) \quad \text{(where } \Psi^+ \text{ irreflexive)} \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \downarrow_{\omega_l} \omega_n]) \quad \text{(where } \Psi^+ \text{ irreflexive)} \]
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.

\[ \Psi; \Gamma \vdash \Sigma P :: (x_s : A_s[\omega_k \downarrow \omega^n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \uparrow \omega^n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

possibly “aliased” linear channels
Manifest deadlock-freedom

\[
\Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \downarrow_{\omega_l} \omega_n]) \quad \text{(where } \Psi^+ \text{ irreflexive)}
\]

\[
\Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow_{\omega_l} \omega_n]) \quad \text{(where } \Psi^+ \text{ irreflexive)}
\]
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \leftrightarrow_{\omega_l} \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \leftrightarrow_{\omega_l} \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

Express invariants A, B, and C in terms of:
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \downarrow_{\omega_l} \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \downarrow_{\omega_l} \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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

\[ \text{min}(\text{parent}) \leq \text{self}(\text{acquired_child}) \leq \text{max}(\text{parent}) \]
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_S : A_S[\omega_k \downarrow_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)} \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow_{\omega_l}]) \quad \text{(where } \Psi^+ \text{ irreflexive)} \]

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

- min(parent) \leq \text{self(acquired_child)} \leq \text{max(parent)}
- max(parent) < \text{min(child)}
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_{S} : A_{S}[\omega_{k} \uparrow_{\omega_{l}}]) \text{ (where } S \text{ is irreflexive)} \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_{L} : A_{L}[\omega_{k} \uparrow_{\omega_{l}}]) \text{ (where } L \text{ is irreflexive)} \]

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

1. \( \min(\text{parent}) \leq \text{self(\text{acquired_child})} \leq \max(\text{parent}) \)
2. \( \max(\text{parent}) < \min(\text{child}) \)

no vertical red arrows
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_s : A_s[\omega_k \uparrow \omega_n]) \text{ (where } \Psi^+ \text{ irreflexive)} \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_l : A_l[\omega_k \downarrow \omega_n]) \text{ (where } \Psi^+ \text{ irreflexive)} \]

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

- \(\text{min}(\text{parent}) \leq \text{self}(\text{acquired\_child}) \leq \text{max}(\text{parent})\)
- \(\text{max}(\text{parent}) < \text{min}(\text{child})\)
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_{\Sigma} P :: (x_S : A_S[\omega_k \downarrow \omega_l]) \] (where \( \Psi^+ \) irreflexive)

\[ \Psi; \Gamma; \Phi; \Delta \vdash_{\Sigma} P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \] (where \( \Psi^+ \) irreflexive)

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

- \( \min(parent) \leq \text{self(acquired\_child)} \leq \max(parent) \)
- \( \max(parent) < \min(child) \)
- for an acquire: lock-up
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_\Sigma P :: (x_S : A_S[\omega_k \downarrow \omega_l]) \] (where irreflexive)

\[ \Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \] (where irreflexive)

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

- \( \text{min}(\text{parent}) \leq \text{self}(\text{acquired}_\text{child}) \leq \text{max}(\text{parent}) \)
- \( \text{max}(\text{parent}) < \text{min}(\text{child}) \)
- for an acquire: lock-up

- no red cycles
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash \Sigma P : (x_S : A_S[\omega_k \rightleftharpoons \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P : (x_L : A_L[\omega_k \rightleftharpoons \omega_n]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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

- \( \min(\text{parent}) \leq \text{self(\text{acquired\_child})} \leq \max(\text{parent}) \)
- \( \max(\text{parent}) < \min(\text{child}) \)
- for an acquire: lock-up
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash_\Sigma P \equiv (x_S : A_S[\omega_k \downarrow_{\omega_L}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash_\Sigma P \equiv (x_L : A_L[\omega_k \uparrow_{\omega_L}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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

- \( \text{min}(\text{parent}) \leq \text{self(} \text{acquired}_\text{child} \text{)} \leq \text{max}(\text{parent}) \)
- \( \text{max}(\text{parent}) < \text{min}(\text{child}) \)
- for an acquire: lock-up
- for right-rule: \( \Phi \) must be empty
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash \Sigma P :: (x_S : A_S[\omega_k \uparrow \omega_l]) \]  
(\text{where } \Psi^+ \text{ irreflexive})

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \downarrow \omega_l]) \]  
(\text{where } \Phi \text{ must be empty})

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

- \( \min(\text{parent}) \leq \text{self(acquired_child)} \leq \max(\text{parent}) \)
- \( \max(\text{parent}) < \min(\text{child}) \)
- for an acquire: lock-up
- for right-rule: \( \Phi \) must be empty

no ingoing red and up-going green arrow
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash \Sigma P :: (x_S : A_S[\omega_k \downarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \downarrow^{\omega_n}_{\omega_l}]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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

- \( \text{min}(\text{parent}) \leq \text{self}\left(\text{acquired\_child}\right) \leq \text{max}(\text{parent}) \)
- \( \text{max}(\text{parent}) < \text{min}(\text{child}) \)
- for an acquire: lock-up
- for right-rule: \( \Phi \) must be empty
Manifest deadlock-freedom

\[ \Psi; \Gamma \vdash \Sigma P :: (x_S : A_S[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

\[ \Psi; \Gamma; \Phi; \Delta \vdash \Sigma P :: (x_L : A_L[\omega_k \uparrow \omega_l]) \quad (\text{where } \Psi^+ \text{ irreflexive}) \]

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

- \( \text{min}(\text{parent}) \leq \text{self}(\text{acquired_child}) \leq \text{max}(\text{parent}) \)
- \( \text{max}(\text{parent}) < \text{min}(\text{child}) \)
- for an acquire: lock-up
- for right-rule: \( \Phi \) 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:
Unique application field for shared session types:

\[
\text{auction} = \uparrow^S_L \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \to \text{money} \to \downarrow^S_L \text{auction}, \\
\text{cancel} : \downarrow^S_L \text{auction} \}, \\
\text{ended} : \text{id} \to \oplus \{ \text{won} : \text{lot} \otimes \downarrow^S_L \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow^S_L \text{auction} \} \}
\]
Unique application field for shared session types:

\[\text{auction} = \uparrow^L_S \oplus \{\text{running} : \& \{\text{bid} : \text{id} \to \text{money} \to \downarrow^L_S \text{auction}, \quad \text{cancel} : \downarrow^L_S \text{auction}\}, \quad \text{ended} : \text{id} \to \oplus \{\text{won} : \text{lot} \otimes \downarrow^L_S \text{auction}, \quad \text{lost} : \text{money} \otimes \downarrow^L_S \text{auction}\}\}\]

Resource analysis for static prediction of execution cost.
Digital contracts (with Hoffmann, Pfenning, and Das)

Unique application field for shared session types:

\[
\text{auction} = \uparrow^S_L \oplus \{ \text{running} : \& \{ \text{bid} : \text{id} \rightarrow \text{money} \rightarrow \downarrow^S_L \text{auction}, \\
\text{cancel} : \downarrow^S_L \text{auction} \}, \\
\text{ended} : \text{id} \rightarrow \oplus \{ \text{won} : \text{lot} \otimes \downarrow^S_L \text{auction}, \\
\text{lost} : \text{money} \otimes \downarrow^S_L \text{auction} \} \}
\]

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.