# PLMW@POPL 2022

- Programming Languages Mentoring Workshop
- Co-located with ACM SIGPLAN Symposium on Principles of Programming Languages
- Program:
  - Research talks
  - Graduate skills talks
  - Panels
  - Mentoring sessions
- Most likely run in hybrid mode

# PLMW@POPL 2022

- Programming Languages Mentoring Workshop
- Co-located with ACM SIGPLAN Symposium on Principles of Programming Languages
- Program:
  - Research talks
  - Graduate skills talks
  - Panels
  - Mentoring sessions
- Most likely run in hybrid mode

Application required. Monitor website for details.

# Session-Typed Concurrent Programming
# Lecture 4

Stephanie Balzer
Carnegie Mellon University

OPLSS 2021
June 26, 2021

# Today's lecture

# Today's lecture

## Recap

- Extended SILL with replication
- Type system and dynamics for the intuitionistic linear and shared session types language $\text{SILL}_S$
- $\text{SILL}_S$ guarantees session fidelity but not deadlock-freedom

# Today's lecture

## Recap

- Extended SILL with replication
- Type system and dynamics for the intuitionistic linear and shared session types language $\text{SILL}_{\text{S}}$
- $\text{SILL}_{\text{S}}$ guarantees session fidelity but not deadlock-freedom

## Next

- Extend $\text{SILL}_{\text{S}}$ with modal worlds to re-establish deadlock-freedom

# Manifest deadlock-freedom

# Recap: manifest sharing

# Recap: manifest sharing

- acquire-release (mutual exclusion):
  - shared processes must be acquired before interaction and released afterwards

# Recap: manifest sharing

- acquire-release (mutual exclusion):
  - shared processes must be acquired before interaction and released afterwards
- equi-synchronizing session type:
  - process must be released back to the same session type at which previously acquired

# Recap: manifest sharing

- acquire-release (mutual exclusion):
  - shared processes must be acquired before interaction and released afterwards
- equi-synchronizing session type:
  - process must be released back to the same session type at which previously acquired

$$\begin{aligned}\mathsf{queue} = \quad & \&\{\mathsf{enq} : \mathsf{int} \rightarrow \quad \mathsf{queue}, \\ & \quad \mathsf{deq} : \oplus\{\mathsf{none} : \quad \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \quad \mathsf{queue}\}\}\end{aligned}$$

# Recap: manifest sharing

- acquire-release (mutual exclusion):
  - shared processes must be acquired before interaction and released afterwards
- equi-synchronizing session type:
  - process must be released back to the same session type at which previously acquired

$$\begin{array}{ll} \mathsf{queue} = \overset{\text{acquire}}{\downarrow} & \&\{\mathsf{enq} : \mathsf{int} \rightarrow \quad \mathsf{queue}, \\ & \quad \mathsf{deq} : \oplus\{\mathsf{none} : \quad \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \quad \mathsf{queue}\}\} \end{array}$$

# Recap: manifest sharing

- acquire-release (mutual exclusion):
  - shared processes must be acquired before interaction and released afterwards
- equi-synchronizing session type:
  - process must be released back to the same session type at which previously acquired

$$\mathsf{queue} = \overset{\text{acquire}}{\downarrow} \& \{\mathsf{enq} : \mathsf{int} \to \overset{\text{release}}{\downarrow} \mathsf{queue},\ \mathsf{deq} : \oplus\{\mathsf{none} : \underset{\text{release}}{\uparrow} \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \overset{\text{release}}{\downarrow} \mathsf{queue}\}\}$$

# Recap: manifest sharing

$$\begin{aligned} \mathsf{queue} = {} & \overset{\text{acquire}}{\downarrow}\ \&\{\mathsf{enq} : \mathsf{int} \to \overset{\text{release}}{\downarrow}\ \mathsf{queue}, \\ & \qquad \mathsf{deq} : \oplus\{\mathsf{none} : \underset{\text{release}}{\uparrow}\ \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \overset{\text{release}}{\downarrow}\ \mathsf{queue}\}\} \end{aligned}$$

# Recap: manifest sharing

Adjoint formulation:

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

$$
\begin{array}{l}
\mathsf{queue} = \overset{\text{acquire}}{\downarrow}\ \&\{\mathsf{enq} : \mathsf{int} \to \overset{\text{release}}{\downarrow}\ \mathsf{queue}, \\
\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \underset{\text{release}}{\uparrow}\ \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \overset{\text{release}}{\downarrow}\ \mathsf{queue}\}\}
\end{array}
$$

# Recap: manifest sharing

Adjoint formulation:

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

acquire release release

$$
\begin{array}{l}
\mathsf{queue} = \;\&\{\mathsf{enq} : \mathsf{int} \rightarrow \mathsf{queue},\\
\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \mathsf{queue}\}\}
\end{array}
$$

release

# Recap: manifest sharing

Adjoint formulation:

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

acquire

release

release

$$
\begin{array}{l}
\mathsf{queue} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{enq} : \mathsf{int} \to \quad \mathsf{queue}, \\
\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \quad \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \quad \mathsf{queue}\}\}
\end{array}
$$

release

# Recap: manifest sharing

Adjoint formulation:

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

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

acquire

release

release

$$\begin{aligned} \mathsf{queue} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{ & \mathsf{enq} : \mathsf{int} \rightarrow \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}, \\ & \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\}\} \end{aligned}$$

release

# Recap: manifest sharing

Adjoint formulation:

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

$$
\begin{aligned}
\mathsf{queue} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{ & \mathsf{enq} : \mathsf{int} \rightarrow \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}, \\
& \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\}\}
\end{aligned}
$$

# Recap: manifest sharing

Adjoint formulation:

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

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

$$\begin{aligned} \mathsf{queue} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{ & \mathsf{enq} : \mathsf{int} \rightarrow \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}, \\ & \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}, \mathsf{some} : \mathsf{int} \times \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\}\} \end{aligned}$$

Take-away: up-arrow = acquire, down-arrow = release

# Recap: manifest sharing

# Recap: manifest sharing

Session type system that allows shared and linear processes to coexist and guarantees:

# Recap: manifest sharing

- Session type system that allows shared and linear processes to coexist and guarantees:
  - data-race-freedom (low-level and high-level)
  - preservation (a.k.a. session fidelity)

# Recap: manifest sharing

- Session type system that allows shared and linear processes to coexist and guarantees:
  - data-race-freedom (low-level and high-level)
  - preservation (a.k.a. session fidelity)
- Implementations:
  - Ferrite DSL in Rust

Ferrite: ferrite-session Rust crate.

# Recap: manifest sharing

- Session type system that allows shared and linear processes to coexist and guarantees:
  - data-race-freedom (low-level and high-level)
  - preservation (a.k.a. session fidelity)
- Implementations:
  - Ferrite DSL in Rust
- Unfortunately, deadlocks are possible now

Ferrite: ferrite-session Rust crate.

# Expressiveness of manifest sharing

# Expressiveness of manifest sharing

→ in simply-typed lambda-calculus, addition of recursive types recovers expressiveness of untyped lambda-calculus

# Expressiveness of manifest sharing

- in simply-typed lambda-calculus, addition of recursive types recovers expressiveness of untyped lambda-calculus
- recursive types are not enough to recover the expressiveness of the untyped pi-calculus for linear session types

# Expressiveness of manifest sharing

- in simply-typed lambda-calculus, addition of recursive types recovers expressiveness of untyped lambda-calculus
- recursive types are not enough to recover the expressiveness of the untyped pi-calculus for linear session types
- both recursive types and shared session types are required

# Expressiveness of manifest sharing

- in simply-typed lambda-calculus, addition of recursive types recovers expressiveness of untyped lambda-calculus
- recursive types are not enough to recover the expressiveness of the untyped pi-calculus for linear session types
- both recursive types and shared session types are required
  - acquire-release introduces nondeterminism

# Expressiveness of manifest sharing

- in simply-typed lambda-calculus, addition of recursive types recovers expressiveness of untyped lambda-calculus
- recursive types are not enough to recover the expressiveness of the untyped pi-calculus for linear session types
- both recursive types and shared session types are required
  - acquire-release introduces nondeterminism
  - encoding of untyped asynchronous pi-calculus into $\text{SILL}_{S}$ and proof of operational and observational correspondence

Stephanie Balzer, Frank Pfenning, and Bernardo Toninho. A Universal Session Type for Untyped Asynchronous Communication. CONCUR 2018.

# Curry-Howard correspondence revisited

# Curry-Howard correspondence revisited

## Linear session types without sharing:

- linear propositions as session types
- proofs as processes
- cut reduction as communication

# Curry-Howard correspondence revisited

## Linear session types without sharing:

- linear propositions as session types
- proofs as processes
- cut reduction as communication

## Manifest sharing:

- correspondence does no longer uphold, but we get:
  - linear propositions as session types
  - proofs as processes
  - interleaving of proof construction (acquire), proof reduction (communication), and proof deconstruction (release)

# Curry-Howard correspondence revisited

## Linear session types without sharing:

- linear propositions as session types
- proofs as processes
- cut reduction as communication

## Manifest sharing:

- correspondence does no longer uphold, but we get:
    - linear propositions as session types
    - proofs as processes
    - interleaving of proof construction (acquire), proof reduction (communication), and proof deconstruction (release)

➡ deadlock: failure of proof construction

# Deadlock example: dining philosophers

# Deadlock example: dining philosophers

not expressible with
linear session types

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$\begin{array}{l} \mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\ c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\ \quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\ \quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\ \quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' \end{array}$$

$$\begin{array}{l} \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\ c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\ \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\ \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\ \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} \end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork}$ $\qquad$ $\mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$\begin{array}{l} thinking : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\ c \leftarrow thinking \leftarrow left, right = \\ \quad left' \leftarrow \mathsf{acquire}\ left\ ; \\ \quad right' \leftarrow \mathsf{acquire}\ right\ ; \\ \quad c \leftarrow eating \leftarrow left', right' \end{array}$$

$$\begin{array}{l} eating : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\ c \leftarrow eating \leftarrow left', right' = \\ \quad right \leftarrow \mathsf{release}\ right'\ ; \\ \quad left \leftarrow \mathsf{release}\ left'\ ; \\ \quad c \leftarrow thinking \leftarrow left, right \end{array}$$

thinking philosopher

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$\begin{array}{l} \mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\ c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\ \quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\ \quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\ \quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' \end{array}$$

$$\begin{array}{l} \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\ c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\ \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\ \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\ \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} \end{array}$$

eating philosopher

# Deadlock example: dining

fork, can be perpetually acquired and released

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$thinking : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\}$

$c \leftarrow thinking \leftarrow left, right =$
$\quad left' \leftarrow \mathsf{acquire}\ left\ ;$
$\quad right' \leftarrow \mathsf{acquire}\ right\ ;$
$\quad c \leftarrow eating \leftarrow left', right'$

$eating : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\}$

$c \leftarrow eating \leftarrow left', right' =$
$\quad right \leftarrow \mathsf{release}\ right'\ ;$
$\quad left \leftarrow \mathsf{release}\ left'\ ;$
$\quad c \leftarrow thinking \leftarrow left, right$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$
\begin{array}{l}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}'
\end{array}
$$

$$
\begin{array}{l}
\mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}
$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad \mathsf{sfork} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$$\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left} \ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}' \ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right} \ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}' \ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$thinking : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\}$$
$$c \leftarrow thinking \leftarrow left, right =$$
$$left' \leftarrow \mathsf{acquire}\ left\ ;$$
$$right' \leftarrow \mathsf{acquire}\ right\ ;$$
$$c \leftarrow eating \leftarrow left', right'$$

$$eating : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\}$$
$$c \leftarrow eating \leftarrow left', right' =$$
$$right \leftarrow \mathsf{release}\ right'\ ;$$
$$left \leftarrow \mathsf{release}\ left'\ ;$$
$$c \leftarrow thinking \leftarrow left, right$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$\begin{array}{l} \mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\ c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\ \quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\ \quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\ \quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' \end{array}$$

$$\begin{array}{l} \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\ c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\ \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\ \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\ \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} \end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$
\begin{array}{l}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}'
\end{array}
$$

$$
\begin{array}{l}
\mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}
$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{lfork}$

$$
\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}
$$

$$
\begin{array}{l}
f_0 \leftarrow \mathit{fork\_proc}\ ;\ f_1 \leftarrow \mathit{fork\_proc}\ ;\ f_2 \leftarrow \mathit{fork\_proc}\ ; \\
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}
$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$$\begin{array}{l}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}'
\end{array}$$

$$\begin{array}{l}
\mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

$$\begin{array}{l}
f_0 \leftarrow \mathit{fork_proc}\ ;\ f_1 \leftarrow \mathit{fork_proc}\ ;\ f_2 \leftarrow \mathit{fork_proc}\ ; \\
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork}$ $\qquad$ $\mathsf{sfork} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$$\begin{array}{l}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}'
\end{array}$$

$$\begin{array}{l}
\mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

$$\begin{array}{l}
f_0 \leftarrow \mathit{fork_proc}\ ;\ f_1 \leftarrow \mathit{fork_proc}\ ;\ f_2 \leftarrow \mathit{fork_proc}\ ; \\
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$$
\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}
$$

$$
\begin{array}{l}
f_0 \leftarrow \mathit{fork\_proc}\ ;\ f_1 \leftarrow \mathit{fork\_proc}\ ;\ f_2 \leftarrow \mathit{fork\_proc}\ ; \\
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}
$$

$$
\begin{array}{c}
p_0 \\
| \\
f_0
\end{array}
$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad \mathsf{sfork} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$$\begin{array}{l}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}'
\end{array}$$

$$\begin{array}{l}
\mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

$$\begin{array}{l}
f_0 \leftarrow \mathit{fork_proc}\ ;\ f_1 \leftarrow \mathit{fork_proc}\ ;\ f_2 \leftarrow \mathit{fork_proc}\ ; \\
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}$$

$$\begin{array}{cc}
\mathsf{p}_0 & \mathsf{p}_1 \\
| & | \\
\mathsf{f}_0 & \mathsf{f}_1
\end{array}$$

# Deadlock example: dining philosophers

$\mathsf{lfork} = {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad\qquad \mathsf{sfork} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$$\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

$$\begin{array}{l}
f_0 \leftarrow \mathit{fork_proc}\ ; f_1 \leftarrow \mathit{fork_proc}\ ; f_2 \leftarrow \mathit{fork_proc}\ ; \\
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}$$

$$\begin{array}{ccc}
p_0 & p_1 & p_2 \\
| & | & | \\
f_0 & f_1 & f_2
\end{array}$$

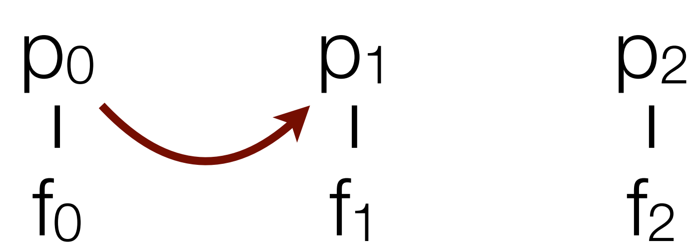
# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{lfork}$

$thinking : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\}$
$c \leftarrow thinking \leftarrow left, right =$
$\quad left' \leftarrow \mathsf{acquire}\ left\ ;$
$\quad right' \leftarrow \mathsf{acquire}\ right\ ;$
$\quad c \leftarrow eating \leftarrow left', right'$

$eating : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\}$
$c \leftarrow eating \leftarrow left', right' =$
$\quad right \leftarrow \mathsf{release}\ right'\ ;$
$\quad left \leftarrow \mathsf{release}\ left'\ ;$
$\quad c \leftarrow thinking \leftarrow left, right$

$f_0 \leftarrow fork\_proc\ ;\ f_1 \leftarrow fork\_proc\ ;\ f_2 \leftarrow fork\_proc\ ;$
$p_0 \leftarrow thinking \leftarrow f_0, f_1\ ;$
$p_1 \leftarrow thinking \leftarrow f_1, f_2\ ;$
$p_2 \leftarrow thinking \leftarrow f_2, f_0\ ;$



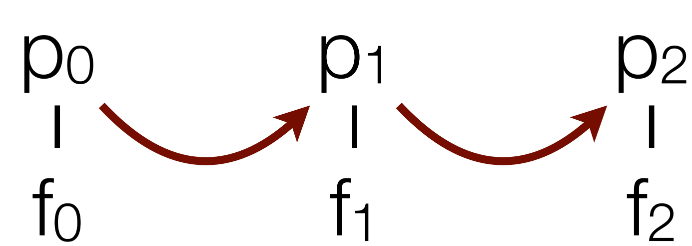
# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}$$

$$f_0 \leftarrow \mathit{fork\_proc}\ ;\ f_1 \leftarrow \mathit{fork\_proc}\ ;\ f_2 \leftarrow \mathit{fork\_proc}\ ;$$

$$\begin{array}{l}
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}$$



# Deadlock example: dining philosophers

$$\mathsf{lfork} = {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{sfork} \qquad\qquad \mathsf{sfork} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$$

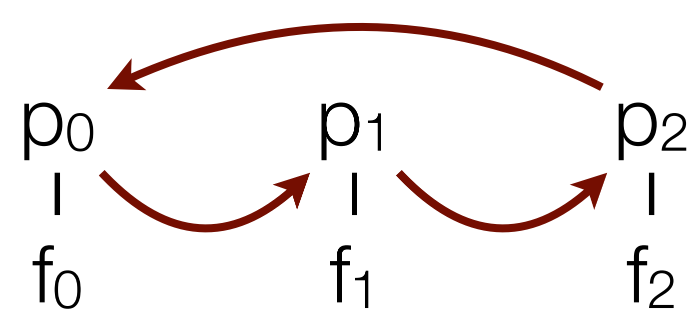
$thinking : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\}$

$c \leftarrow thinking \leftarrow left, right =$
$\quad left' \leftarrow \mathsf{acquire}\ left\ ;$
$\quad right' \leftarrow \mathsf{acquire}\ right\ ;$
$\quad c \leftarrow eating \leftarrow left', right'$

$eating : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\}$

$c \leftarrow eating \leftarrow left', right' =$
$\quad right \leftarrow \mathsf{release}\ right'\ ;$
$\quad left \leftarrow \mathsf{release}\ left'\ ;$
$\quad c \leftarrow thinking \leftarrow left, right$

$f_0 \leftarrow fork\_proc\ ;\ f_1 \leftarrow fork\_proc\ ;\ f_2 \leftarrow fork\_proc\ ;$
$p_0 \leftarrow thinking \leftarrow f_0, f_1\ ;$
$p_1 \leftarrow thinking \leftarrow f_1, f_2\ ;$
$p_2 \leftarrow thinking \leftarrow f_2, f_0\ ;$



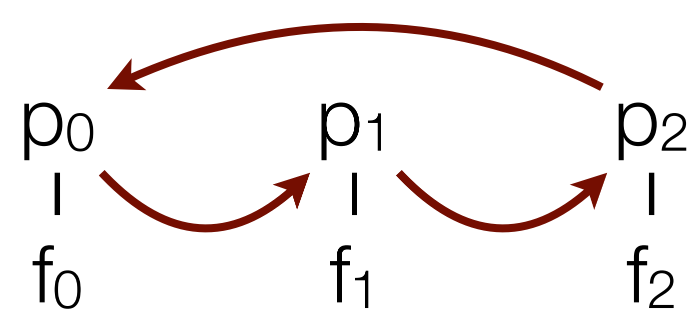
# Deadlock example: dining philosophers

$\mathsf{lfork} = \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{sfork} \qquad \mathsf{sfork} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{lfork}$

$$
\begin{array}{ll}
\mathit{thinking} : \{\mathsf{phil} \leftarrow \mathsf{sfork}, \mathsf{sfork}\} & \mathit{eating} : \{\mathsf{phil} \leftarrow \mathsf{lfork}, \mathsf{lfork}\} \\
c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right} = & c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' = \\
\quad \mathit{left}' \leftarrow \mathsf{acquire}\ \mathit{left}\ ; & \quad \mathit{right} \leftarrow \mathsf{release}\ \mathit{right}'\ ; \\
\quad \mathit{right}' \leftarrow \mathsf{acquire}\ \mathit{right}\ ; & \quad \mathit{left} \leftarrow \mathsf{release}\ \mathit{left}'\ ; \\
\quad c \leftarrow \mathit{eating} \leftarrow \mathit{left}', \mathit{right}' & \quad c \leftarrow \mathit{thinking} \leftarrow \mathit{left}, \mathit{right}
\end{array}
$$

$$f_0 \leftarrow \mathit{fork\_proc}\ ;\ f_1 \leftarrow \mathit{fork\_proc}\ ;\ f_2 \leftarrow \mathit{fork\_proc}\ ;$$

$$
\begin{array}{l}
p_0 \leftarrow \mathit{thinking} \leftarrow f_0, f_1\ ; \\
p_1 \leftarrow \mathit{thinking} \leftarrow f_1, f_2\ ; \\
p_2 \leftarrow \mathit{thinking} \leftarrow f_2, f_0\ ;
\end{array}
$$




deadlock due to cyclic acquisitions

# Another deadlock example

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \textsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$

$\quad c \leftarrow contester \leftarrow sr \;;$

$\quad lr \leftarrow \textsf{acquire}\ sr \;;$

$\quad \textsf{case}\ c\ \textsf{of}$

$\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \;;$

$\qquad\qquad sr \leftarrow \textsf{release}\ lr \;;$

$\qquad\qquad \textsf{close}\ o$

$contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$

$\quad lr \leftarrow \textsf{acquire}\ sr \;;$

$\quad c.\textsf{ping} \;;$

$\quad sr \leftarrow \textsf{release}\ lr \;;$

$\quad \textsf{close}\ c$

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \textsf{sres}\}$
$o \leftarrow owner \leftarrow sr =$
$\quad c \leftarrow contester \leftarrow sr\ ;$
$\quad lr \leftarrow \textsf{acquire}\ sr\ ;$
$\quad \textsf{case}\ c\ \textsf{of}$
$\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c\ ;$
$\qquad\qquad sr \leftarrow \textsf{release}\ lr\ ;$
$\qquad\qquad \textsf{close}\ o$

$contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\}$
$c \leftarrow contester \leftarrow sr =$
$\quad lr \leftarrow \textsf{acquire}\ sr\ ;$
$\quad c.\textsf{ping}\ ;$
$\quad sr \leftarrow \textsf{release}\ lr\ ;$
$\quad \textsf{close}\ c$

# Another deadlock example

$$
\begin{array}{l}
owner : \{\mathbf{1} \leftarrow \boxed{\mathsf{sres}}\} \\
o \leftarrow owner \leftarrow \boxed{sr} = \\
\quad c \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \mathsf{release}\ lr \ ; \\
\qquad\qquad\quad \mathsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \boxed{\mathsf{sres}}\} \\
c \leftarrow contester \leftarrow \boxed{sr} = \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad c.\mathsf{ping} \ ; \\
\quad sr \leftarrow \mathsf{release}\ lr \ ; \\
\quad \mathsf{close}\ c
\end{array}
$$

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \mathsf{sres}\}$
$o \leftarrow owner \leftarrow sr =$
$\quad c \leftarrow contester \leftarrow sr$ ;
$\quad lr \leftarrow$ acquire $sr$ ;
$\quad$ case $c$ of
$\quad \mid$ ping $\rightarrow$ wait $c$ ;
$\qquad\qquad sr \leftarrow$ release $lr$ ;
$\qquad\qquad$ close $o$

$contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\}$
$c \leftarrow contester \leftarrow sr =$
$\quad lr \leftarrow$ acquire $sr$ ;
$\quad c$.ping ;
$\quad sr \leftarrow$ release $lr$ ;
$\quad$ close $c$

# Another deadlock example

$$
\begin{array}{l}
owner : \{\mathbf{1} \leftarrow \textsf{sres}\} \\
o \leftarrow owner \leftarrow sr = \\
\quad c \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad \textsf{case}\ c\ \textsf{of} \\
\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \textsf{release}\ lr \ ; \\
\qquad\qquad\quad \textsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\} \\
c \leftarrow contester \leftarrow sr = \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad c.\textsf{ping} \ ; \\
\quad sr \leftarrow \textsf{release}\ lr \ ; \\
\quad \textsf{close}\ c
\end{array}
$$

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \mathsf{sres}\}$
$o \leftarrow owner \leftarrow sr =$
$\quad c \leftarrow contester \leftarrow sr \ ;$
$\quad lr \leftarrow \mathsf{acquire}\ sr \ ;$
$\quad \mathsf{case}\ c\ \mathsf{of}$
$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ;$
$\qquad\qquad sr \leftarrow \mathsf{release}\ lr \ ;$
$\qquad\qquad \mathsf{close}\ o$

$contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\}$
$c \leftarrow contester \leftarrow sr =$
$\quad lr \leftarrow \mathsf{acquire}\ sr \ ;$
$\quad c.\mathsf{ping} \ ;$
$\quad sr \leftarrow \mathsf{release}\ lr \ ;$
$\quad \mathsf{close}\ c$

# Another deadlock example

$$
\begin{array}{l}
owner : \{\mathbf{1} \leftarrow \textsf{sres}\} \\
o \leftarrow owner \leftarrow sr = \\
\quad c \leftarrow contester \leftarrow sr \ ; \\
\quad \boxed{lr \leftarrow \textsf{acquire}\ sr \ ;} \\
\quad \textsf{case}\ c\ \textsf{of} \\
\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \textsf{release}\ lr \ ; \\
\qquad\qquad\quad \textsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\} \\
c \leftarrow contester \leftarrow sr = \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad c.\textsf{ping} \ ; \\
\quad sr \leftarrow \textsf{release}\ lr \ ; \\
\quad \textsf{close}\ c
\end{array}
$$

# Another deadlock example

$$
\begin{array}{l}
owner : \{\mathbf{1} \leftarrow \mathsf{sres}\} \\
o \leftarrow owner \leftarrow sr = \\
\quad c \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \mathsf{release}\ lr \ ; \\
\qquad\qquad\quad \mathsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\} \\
c \leftarrow contester \leftarrow sr = \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad c.\mathsf{ping} \ ; \\
\quad sr \leftarrow \mathsf{release}\ lr \ ; \\
\quad \mathsf{close}\ c
\end{array}
$$

# Another deadlock example

$$
\begin{array}{l}
owner : \{\mathbf{1} \leftarrow \textsf{sres}\} \\
o \leftarrow owner \leftarrow sr = \\
\quad c \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad \textsf{case}\ c\ \textsf{of} \\
\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \textsf{release}\ lr \ ; \\
\qquad\qquad\quad \textsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\} \\
c \leftarrow contester \leftarrow sr = \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad c.\textsf{ping} \ ; \\
\quad sr \leftarrow \textsf{release}\ lr \ ; \\
\quad \textsf{close}\ c
\end{array}
$$

# Another deadlock example

$$
\begin{array}{l}
\mathit{owner} : \{\mathbf{1} \leftarrow \mathsf{sres}\} \\
o \leftarrow \mathit{owner} \leftarrow sr = \\
\quad c \leftarrow \mathit{contester} \leftarrow sr \; ; \\
\quad lr \leftarrow \mathsf{acquire}\; sr \; ; \\
\quad \mathsf{case}\; c \;\mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\; c \; ; \\
\qquad\qquad sr \leftarrow \mathsf{release}\; lr \; ; \\
\qquad\qquad \mathsf{close}\; o
\end{array}
$$

$$
\begin{array}{l}
\mathit{contester} : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\} \\
c \leftarrow \mathit{contester} \leftarrow sr = \\
\quad lr \leftarrow \mathsf{acquire}\; sr \; ; \\
\quad \boxed{c.\mathsf{ping} \; ;} \\
\quad sr \leftarrow \mathsf{release}\; lr \; ; \\
\quad \mathsf{close}\; c
\end{array}
$$

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \mathsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$
$\quad c \leftarrow contester \leftarrow sr\ ;$
$\quad lr \leftarrow \mathsf{acquire}\ sr\ ;$
$\quad \mathsf{case}\ c\ \mathsf{of}$
$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c\ ;$
$\qquad\qquad sr \leftarrow \mathsf{release}\ lr\ ;$
$\qquad\qquad \mathsf{close}\ o$

$contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$
$\quad lr \leftarrow \mathsf{acquire}\ sr\ ;$
$\quad c.\mathsf{ping}\ ;$
$\quad sr \leftarrow \mathsf{release}\ lr\ ;$
$\quad \mathsf{close}\ c$

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \textsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$

$\quad c \leftarrow contester \leftarrow sr$ ;

$\quad lr \leftarrow \textsf{acquire}\ sr$ ;

$\quad \textsf{case}\ c\ \textsf{of}$

$\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c$ ;

$\qquad\qquad sr \leftarrow \textsf{release}\ lr$ ;

$\qquad\qquad \textsf{close}\ o$

$contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$

$\quad lr \leftarrow \textsf{acquire}\ sr$ ;

$\quad c.\textsf{ping}$ ;

$\quad sr \leftarrow \textsf{release}\ lr$ ;

$\quad \textsf{close}\ c$

$r_0 \leftarrow res\_proc$ ;

$o_0 \leftarrow owner \leftarrow r_0$ ;

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \mathsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$

$\quad c \leftarrow contester \leftarrow sr \;;$

$\quad lr \leftarrow \mathsf{acquire}\ sr \;;$

$\quad \mathsf{case}\ c\ \mathsf{of}$

$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \;;$

$\qquad\qquad sr \leftarrow \mathsf{release}\ lr \;;$

$\qquad\qquad \mathsf{close}\ o$

$contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$

$\quad lr \leftarrow \mathsf{acquire}\ sr \;;$

$\quad c.\mathsf{ping} \;;$

$\quad sr \leftarrow \mathsf{release}\ lr \;;$

$\quad \mathsf{close}\ c$

$r_0 \leftarrow res\_proc \;;$

$o_0 \leftarrow owner \leftarrow r_0 \;;$

$o_0$ $\quad$ $c_0$

|

$r_0$

# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \textsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$

$\quad c \leftarrow contester \leftarrow sr \; ;$

$\quad lr \leftarrow \textsf{acquire}\ sr \; ;$

$\quad \textsf{case}\ c\ \textsf{of}$

$\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \; ;$

$\qquad\qquad sr \leftarrow \textsf{release}\ lr \; ;$

$\qquad\qquad \textsf{close}\ o$

$contester : \{\oplus\{\textsf{ping} : \mathbf{1}\} \leftarrow \textsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$

$\quad lr \leftarrow \textsf{acquire}\ sr \; ;$

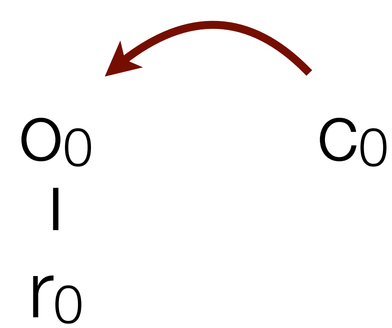
$\quad c.\textsf{ping} \; ;$

$\quad sr \leftarrow \textsf{release}\ lr \; ;$

$\quad \textsf{close}\ c$

$r_0 \leftarrow res\_proc \; ;$

$o_0 \leftarrow owner \leftarrow r_0 \; ;$



# Another deadlock example

$owner : \{\mathbf{1} \leftarrow \mathsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$

$\quad c \leftarrow contester \leftarrow sr \;;$

$\quad lr \leftarrow \mathsf{acquire}\; sr \;;$

$\quad \mathsf{case}\; c\; \mathsf{of}$

$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\; c \;;$

$\qquad\qquad sr \leftarrow \mathsf{release}\; lr \;;$

$\qquad\qquad \mathsf{close}\; o$

$contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$

$\quad lr \leftarrow \mathsf{acquire}\; sr \;;$

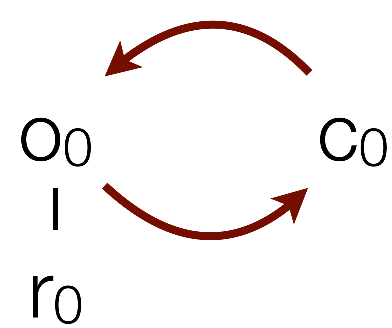
$\quad c.\mathsf{ping} \;;$

$\quad sr \leftarrow \mathsf{release}\; lr \;;$

$\quad \mathsf{close}\; c$

$r_0 \leftarrow res\_proc \;;$

$o_0 \leftarrow owner \leftarrow r_0 \;;$



# Another deadlock example

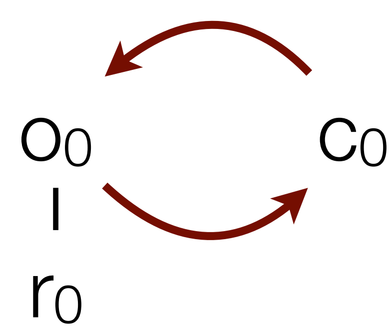
$owner : \{\mathbf{1} \leftarrow \mathsf{sres}\}$

$o \leftarrow owner \leftarrow sr =$
$\quad c \leftarrow contester \leftarrow sr \; ;$
$\quad lr \leftarrow \mathsf{acquire}\; sr \; ;$
$\quad \mathsf{case}\; c \;\mathsf{of}$
$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\; c \; ;$
$\qquad\qquad sr \leftarrow \mathsf{release}\; lr \; ;$
$\qquad\qquad \mathsf{close}\; o$

$contester : \{\oplus\{\mathsf{ping} : \mathbf{1}\} \leftarrow \mathsf{sres}\}$

$c \leftarrow contester \leftarrow sr =$
$\quad lr \leftarrow \mathsf{acquire}\; sr \; ;$
$\quad c.\mathsf{ping} \; ;$
$\quad sr \leftarrow \mathsf{release}\; lr \; ;$
$\quad \mathsf{close}\; c$

$r_0 \leftarrow res\_proc \; ;$
$o_0 \leftarrow owner \leftarrow r_0 \; ;$




deadlock due interdependent acquisitions and synchronizations

# Another deadlock example

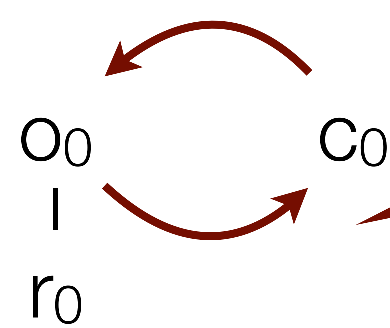
$owner : \{\mathbf{1} \leftarrow \text{sres}\}$

$o \leftarrow owner \leftarrow sr =$
$\quad c \leftarrow contester \leftarrow sr\ ;$
$\quad lr \leftarrow \text{acquire } sr\ ;$
$\quad \text{case } c \text{ of}$
$\quad \mid \text{ping} \rightarrow \text{wait } c\ ;$
$\qquad\qquad sr \leftarrow \text{release } lr\ ;$
$\qquad\qquad \text{close } o$

$contester : \{\oplus\{\text{ping} : \mathbf{1}\} \leftarrow \text{sres}\}$

$c \leftarrow contester \leftarrow sr =$
$\quad lr \leftarrow \text{acquire } sr\ ;$
$\quad c.\text{ping}\ ;$
$\quad sr \leftarrow \text{release } lr\ ;$
$\quad \text{close } c$

$r_0 \leftarrow res\_proc\ ;$
$o_0 \leftarrow owner \leftarrow r_0\ ;$




deadlock
may occur both in
synchronous and
asynchronous
semantics

deadlock due interdependent acquisitions and synchronizations

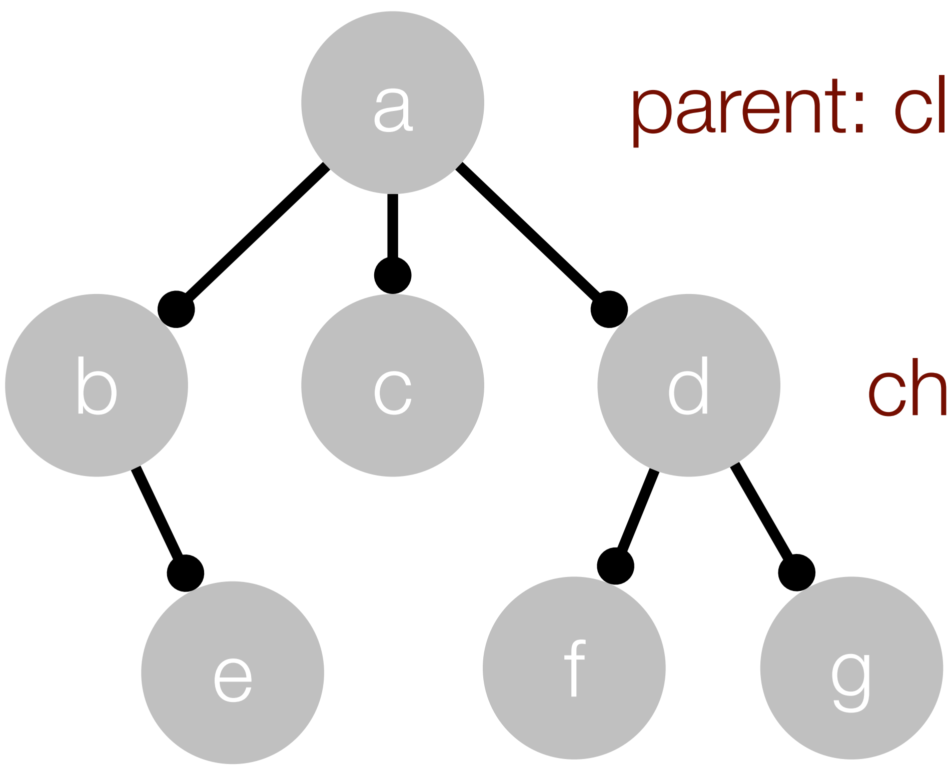
# Why are linear session types deadlock-free?

# 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.



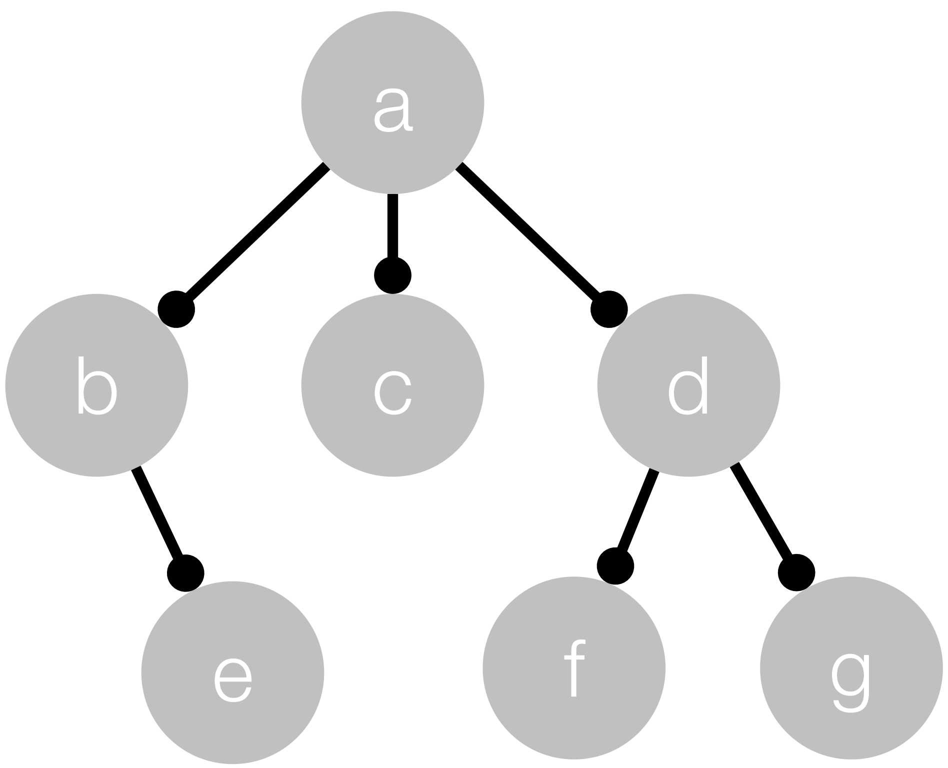

parent: client

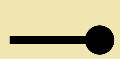
child: provider

**Legend:** linear channel

# Why are linear session types deadlock-free?

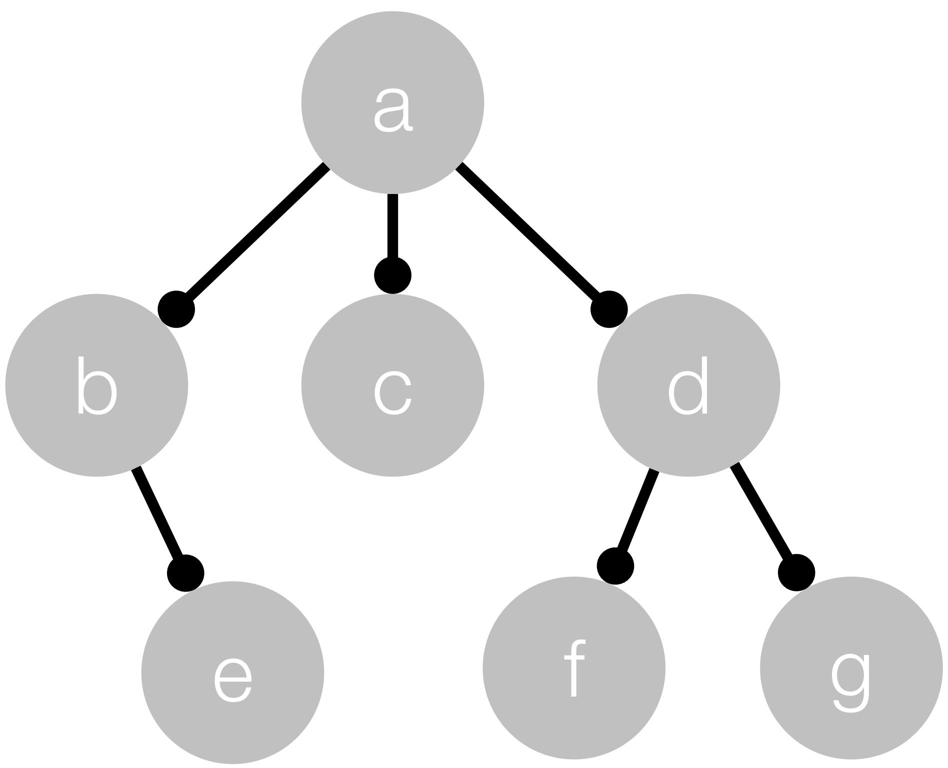
Linearity ("exactly one client") turns process graph into a tree.



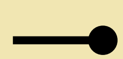

**Legend:**  linear channel

# Why are linear session types deadlock-free?

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

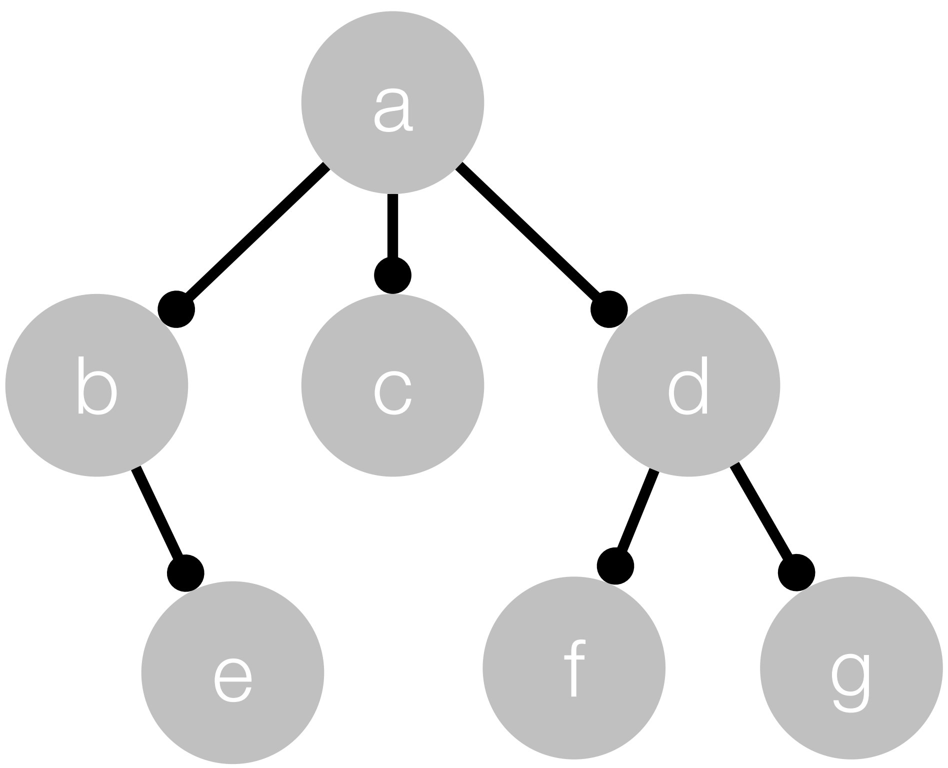



What are the threats to progress?



# Why are linear session types deadlock-free?

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



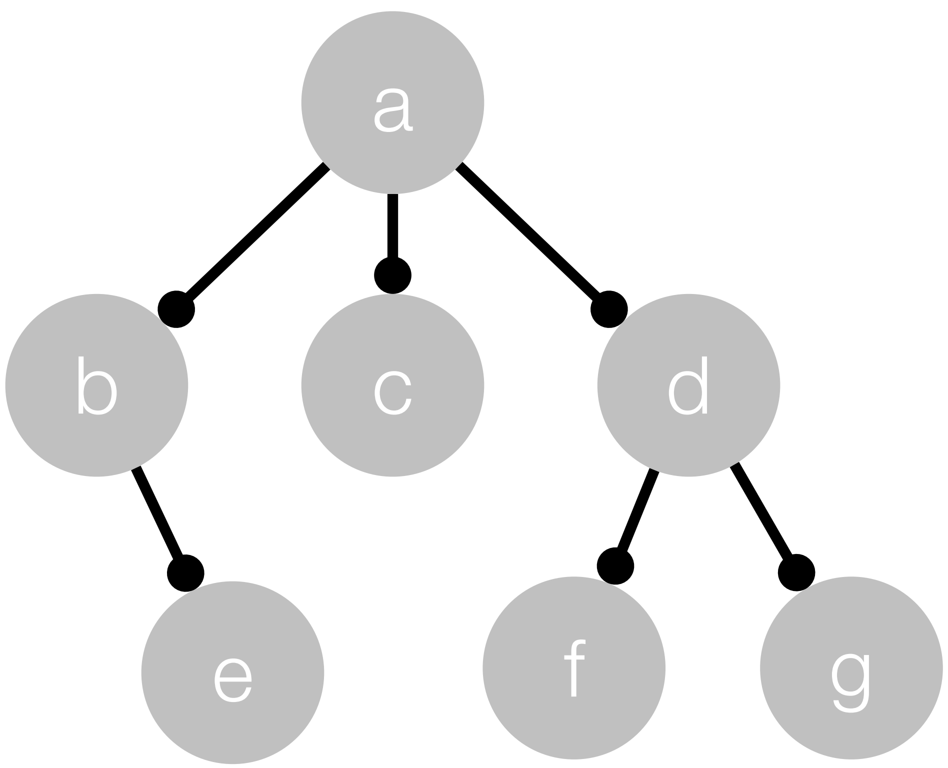

What are the threats to progress?

- Two scenarios:

**Legend:** linear channel

# Why are linear session types deadlock-free?

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



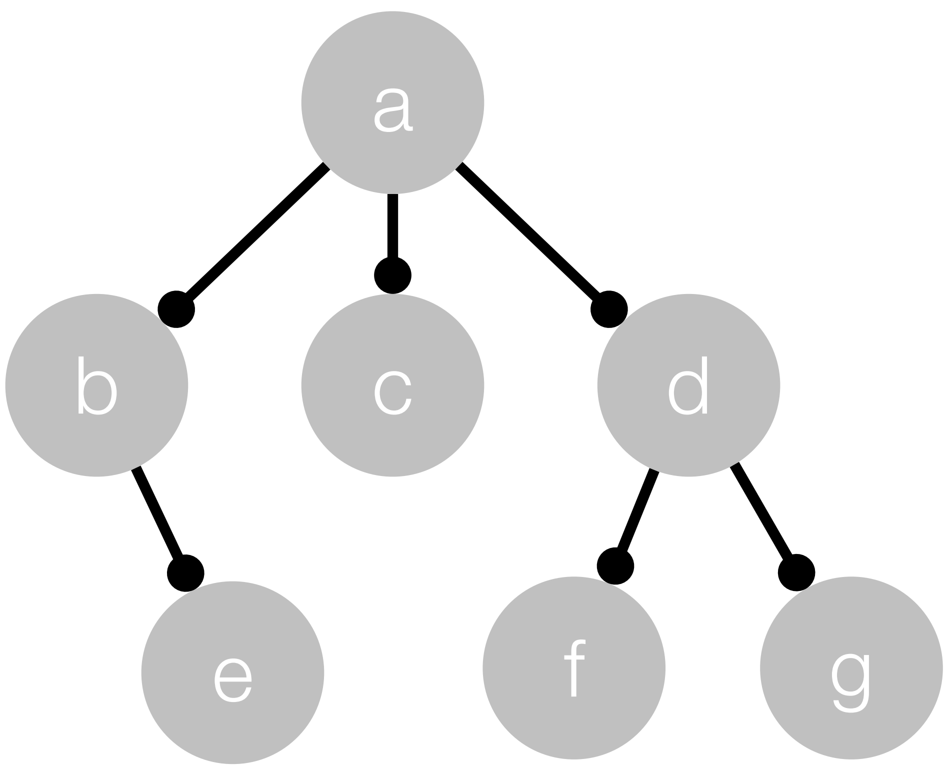

What are the threats to progress?

- 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.



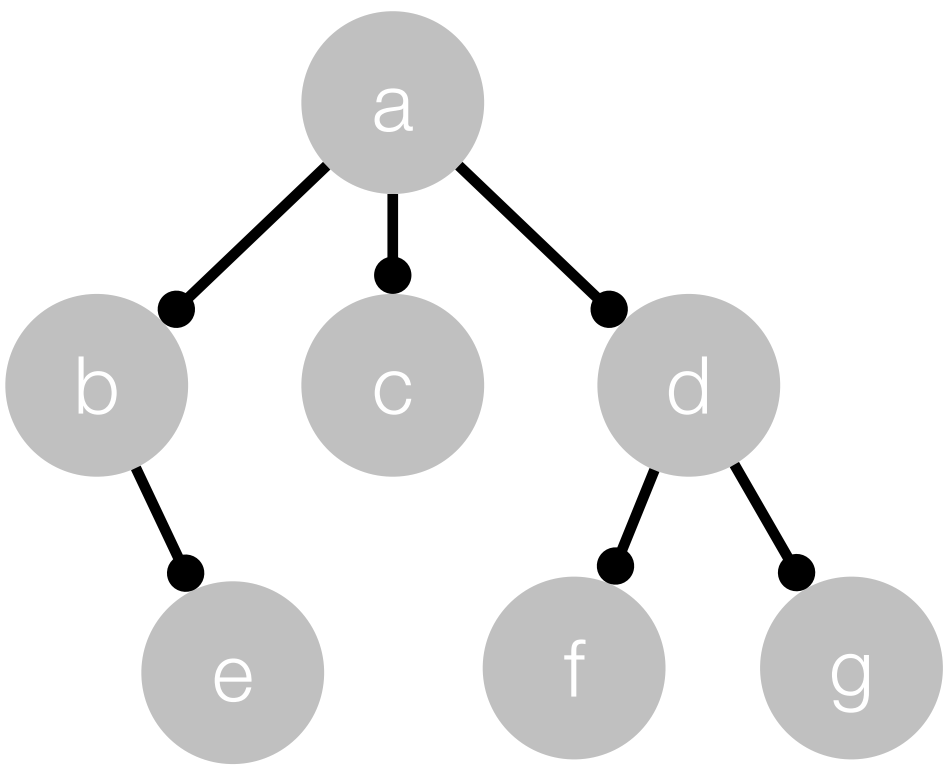

What are the threats to progress?

- 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.



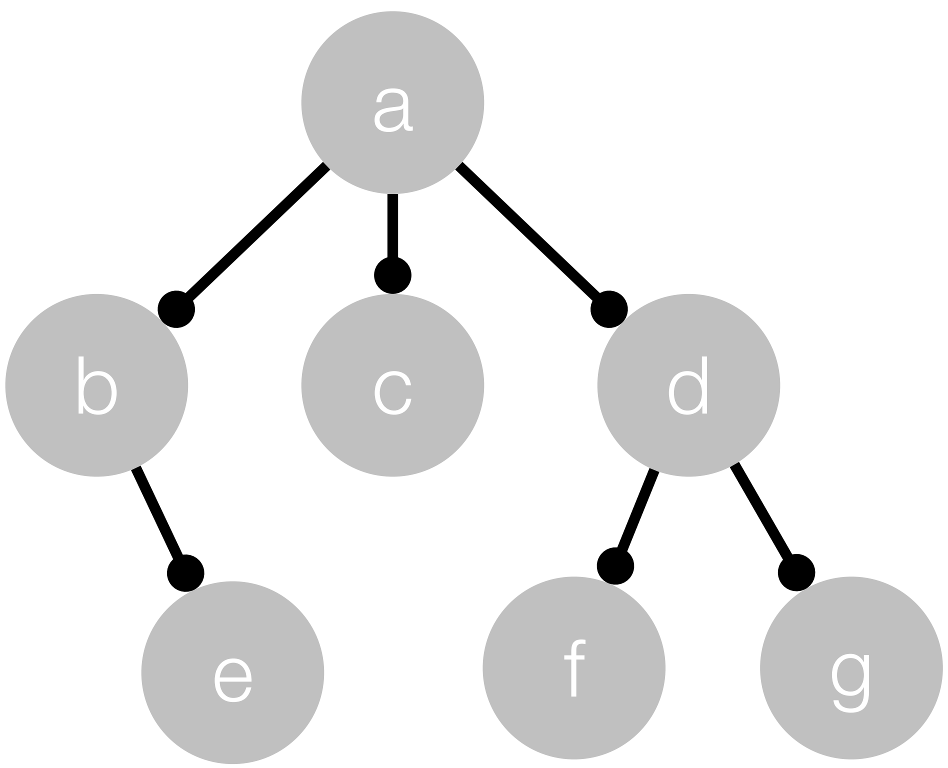

## What are the threats to progress?

- 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?

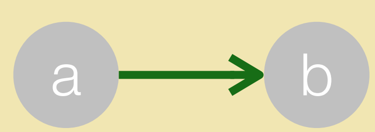
Linearity (“exactly one client”) turns process graph into a tree.




## 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

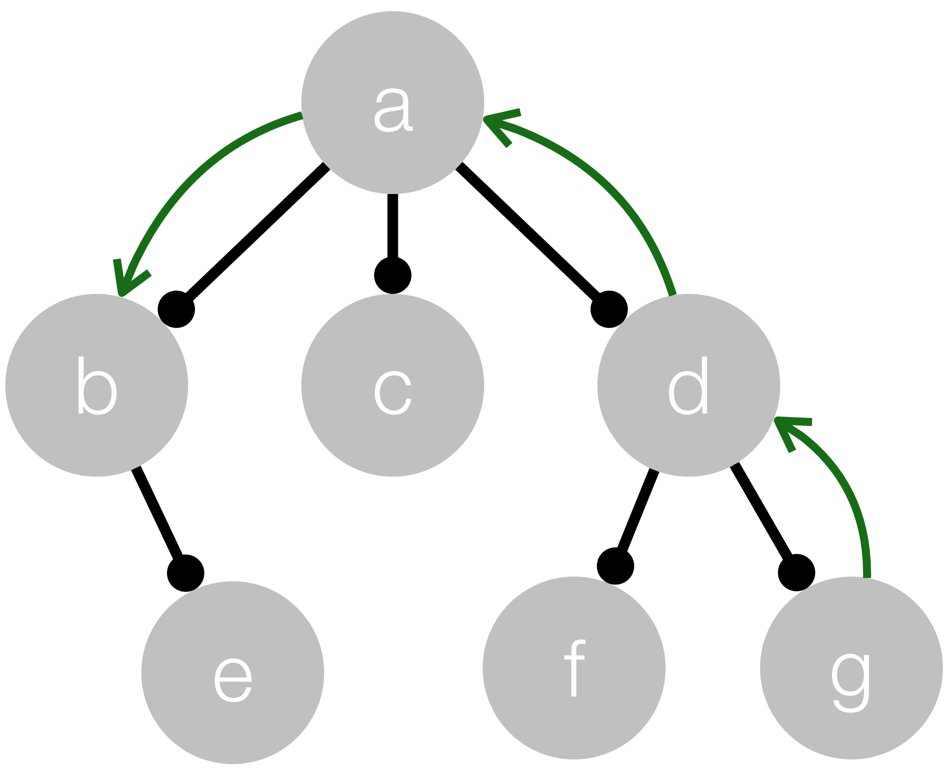
**Legend:** linear channel




“a waits for b”

# Why are linear session types deadlock-free?

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



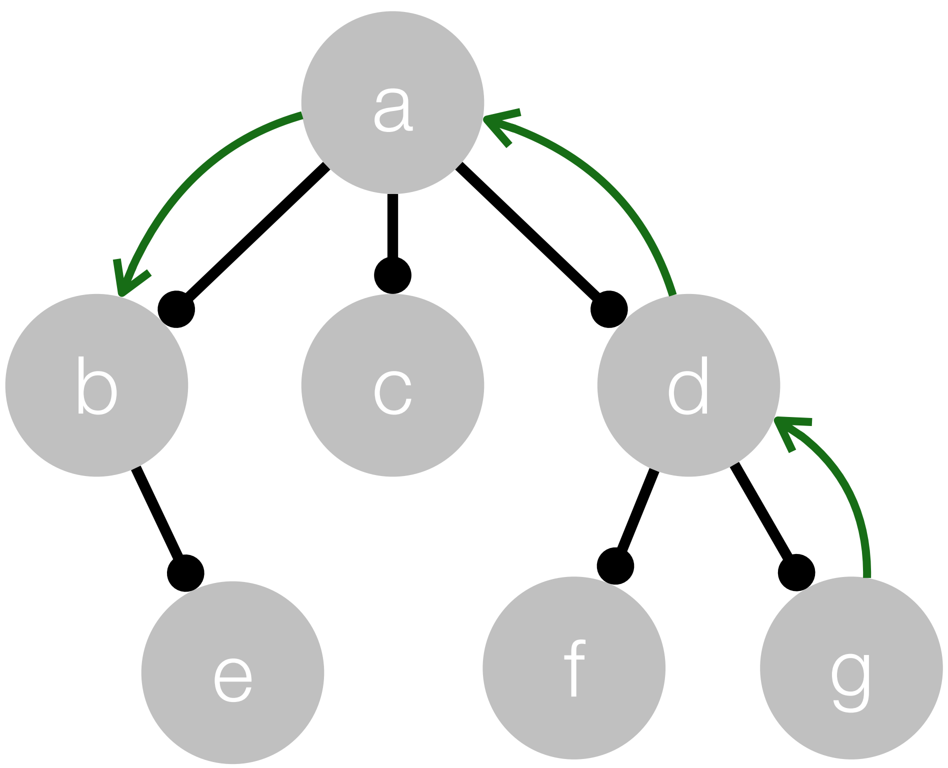

## What are the threats to progress?

- 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 → b “a waits for b”

# Why are linear session types deadlock-free?

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

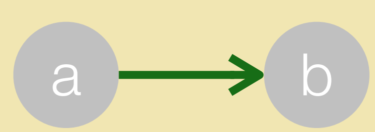



What are the threats to progress?

- 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 waits for b”

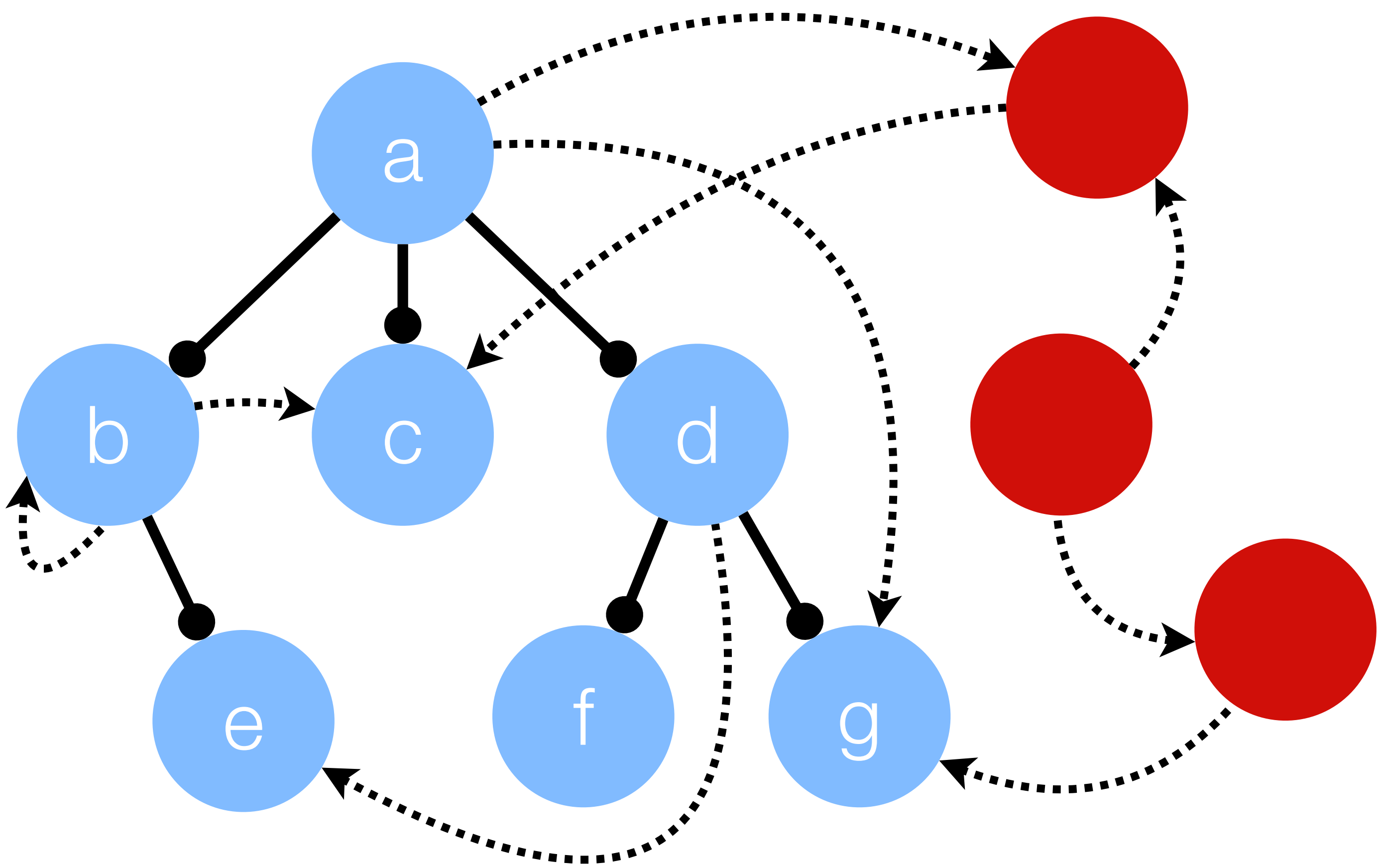
# Let’s add sharing

# Let’s add sharing

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

# Let's add sharing

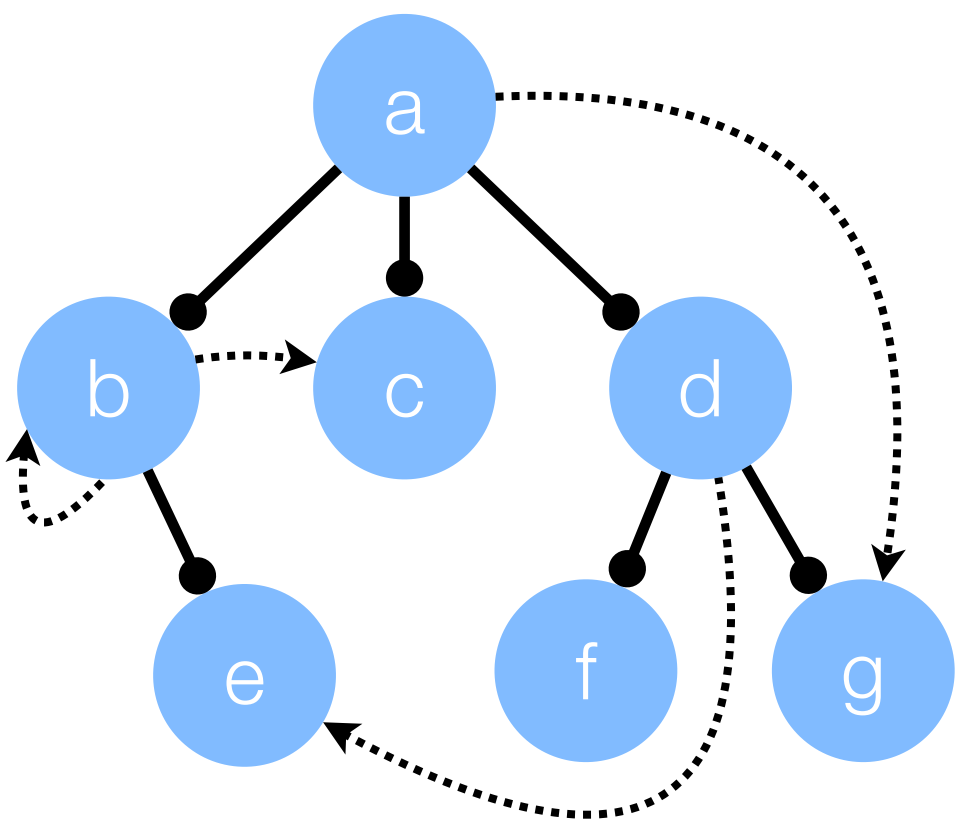
We get a graph of linear and shared processes, with a linear tree inside.




**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.

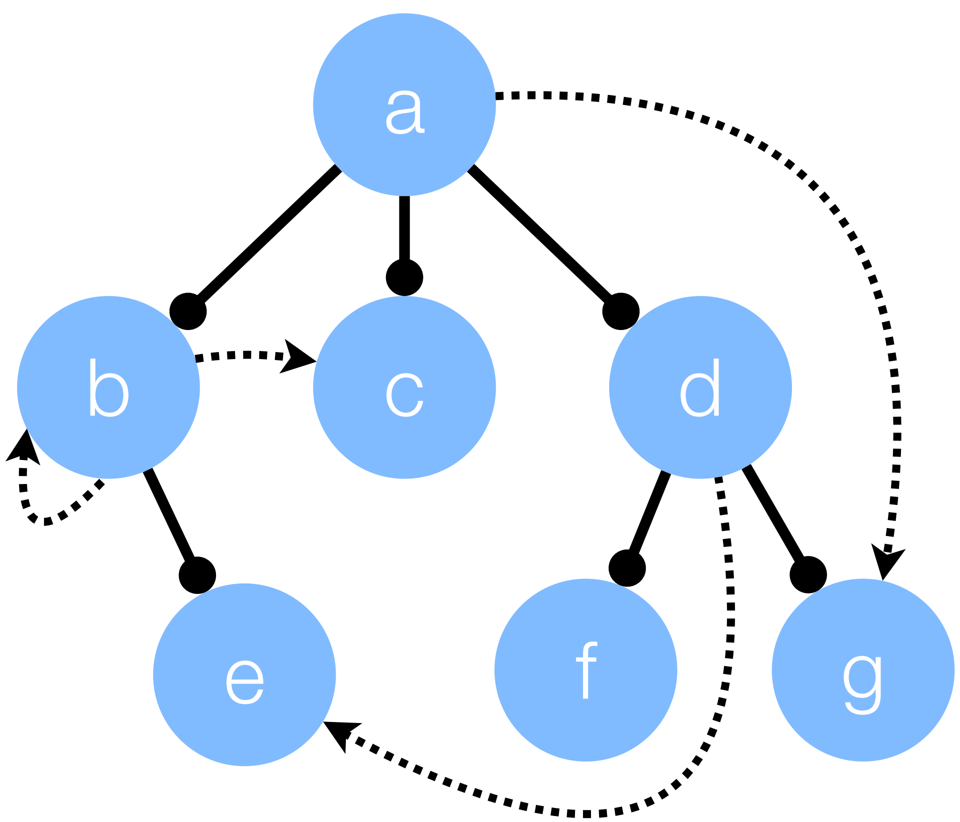



**Legend:** linear channel linear process shared process

shared channel

# Let's add sharing

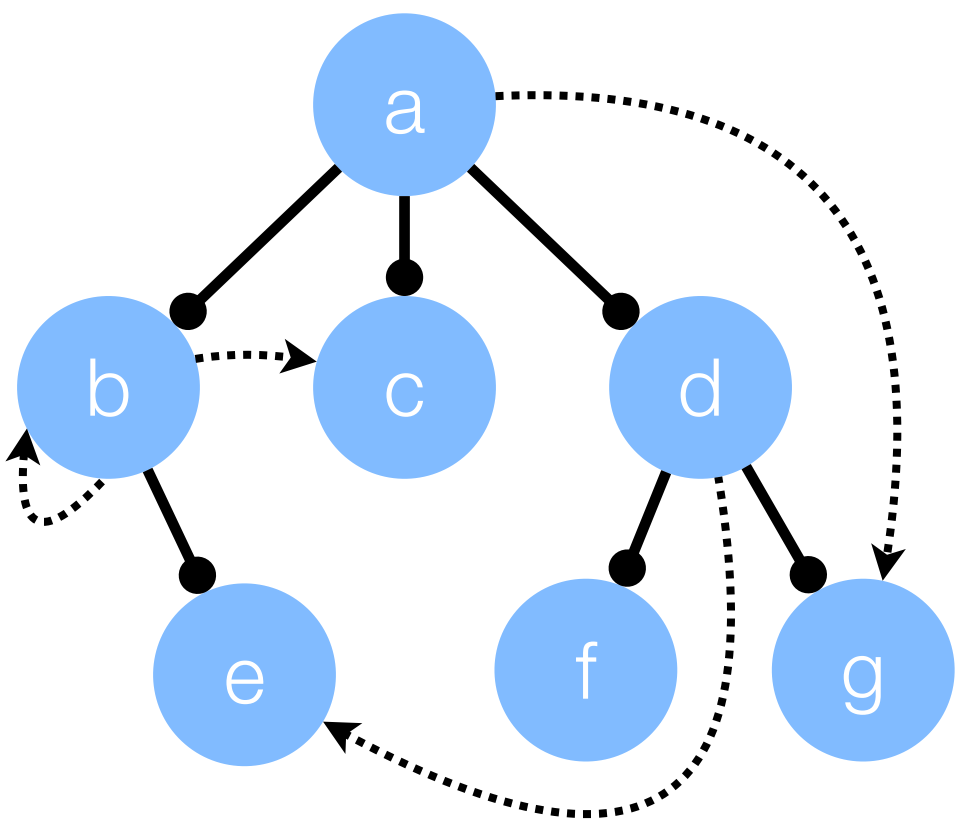
We get a graph of linear and shared processes, with a linear tree inside.




**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.



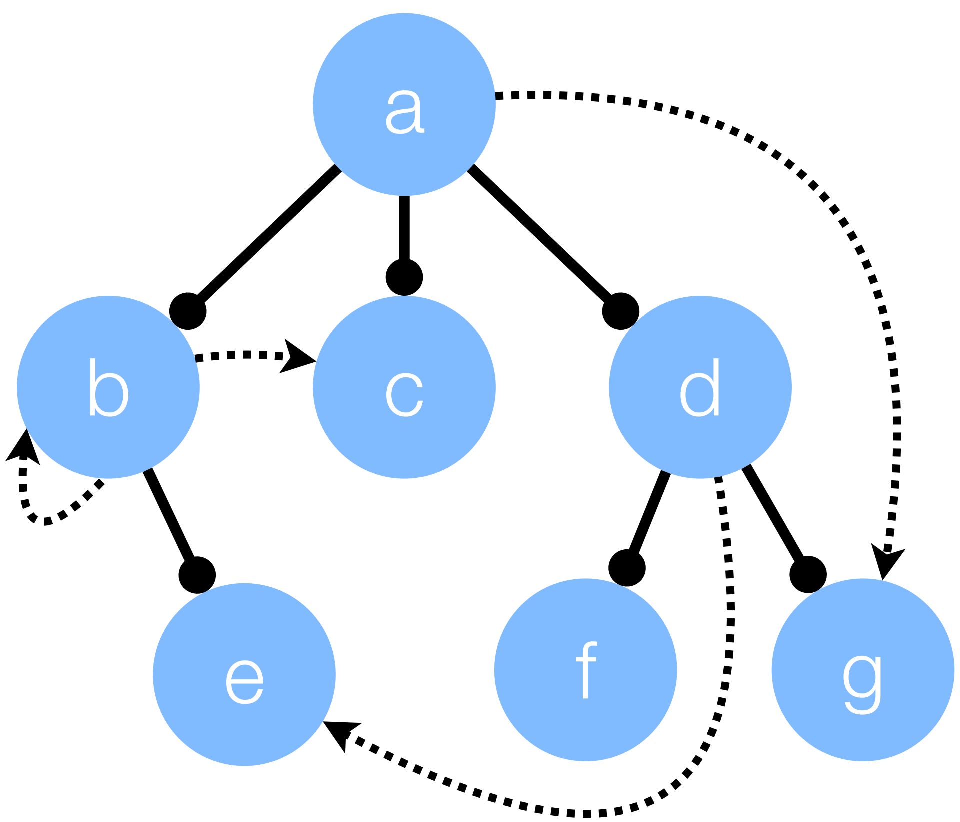

Acquire-release amounts to “locking”

**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”

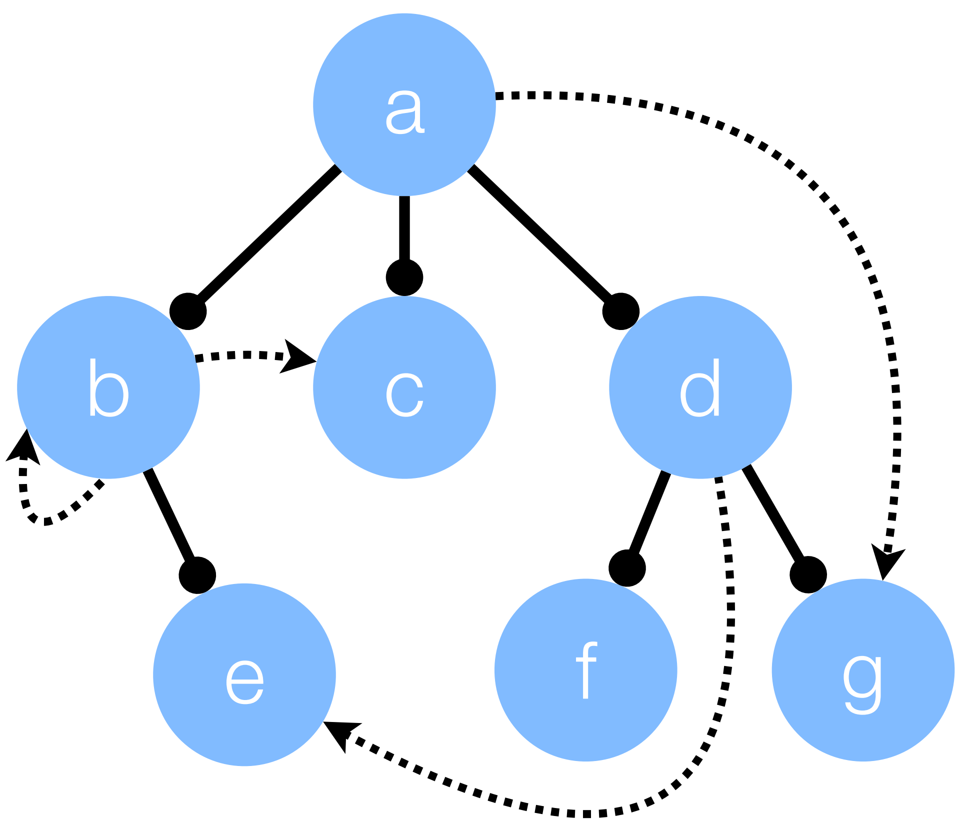
- Possibility of cyclic dependencies

**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”

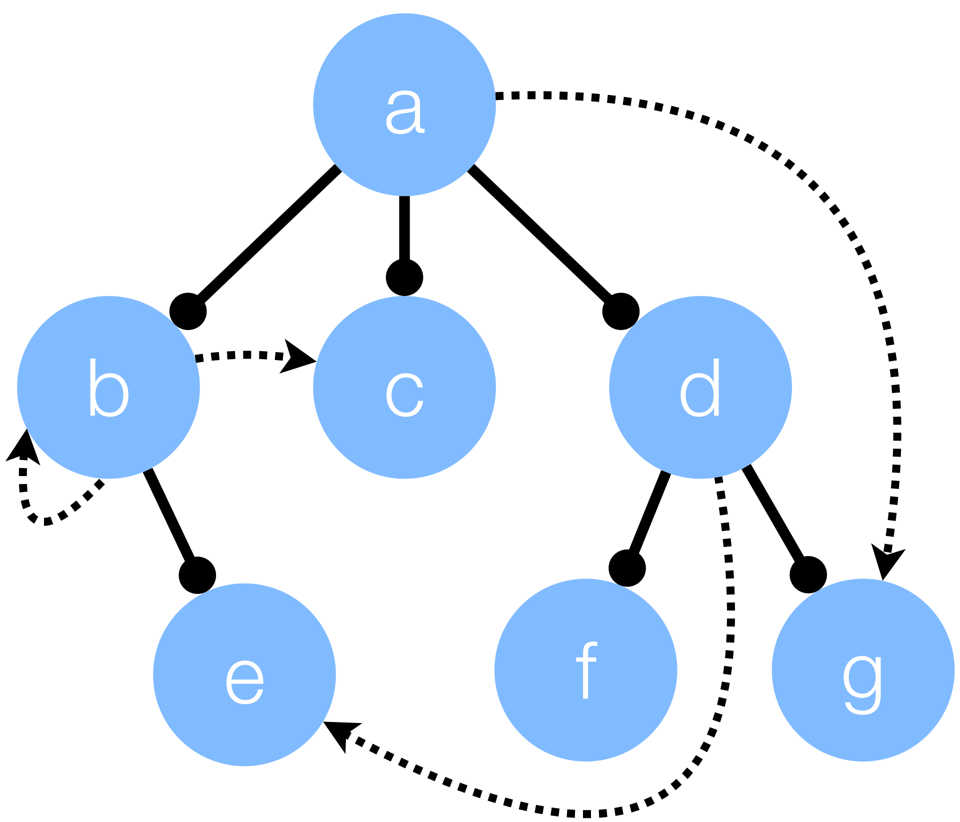
- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow

**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.



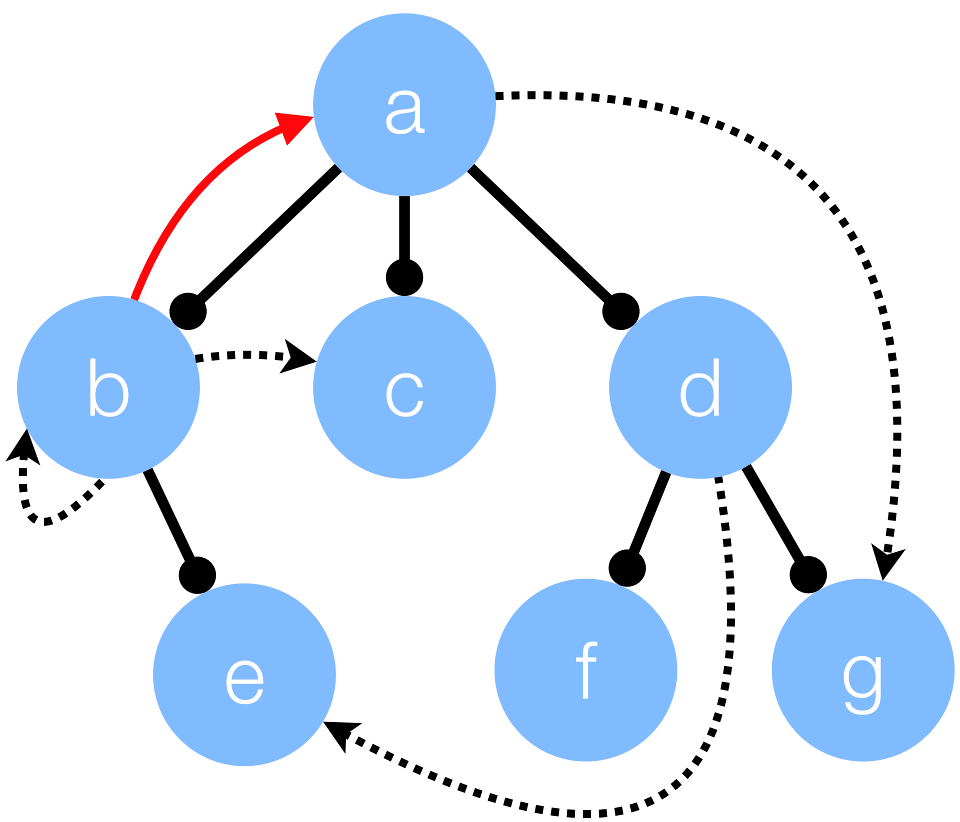

Acquire-release amounts to “locking”

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

**Legend:** linear channel · 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”

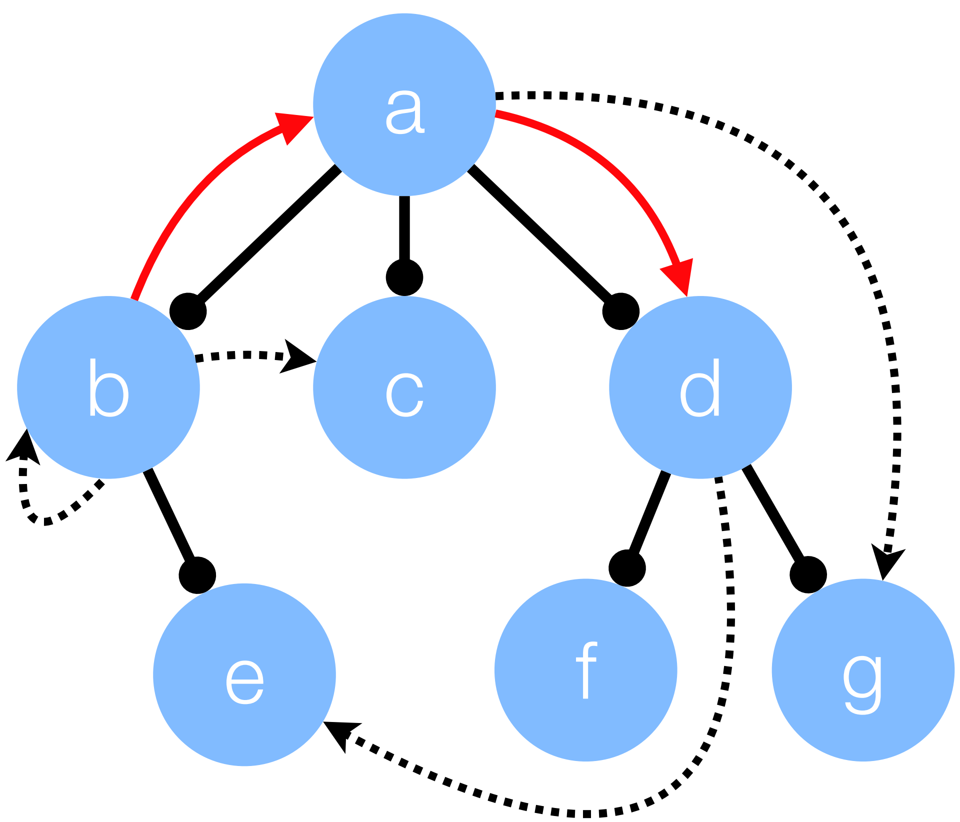
- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow

**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”

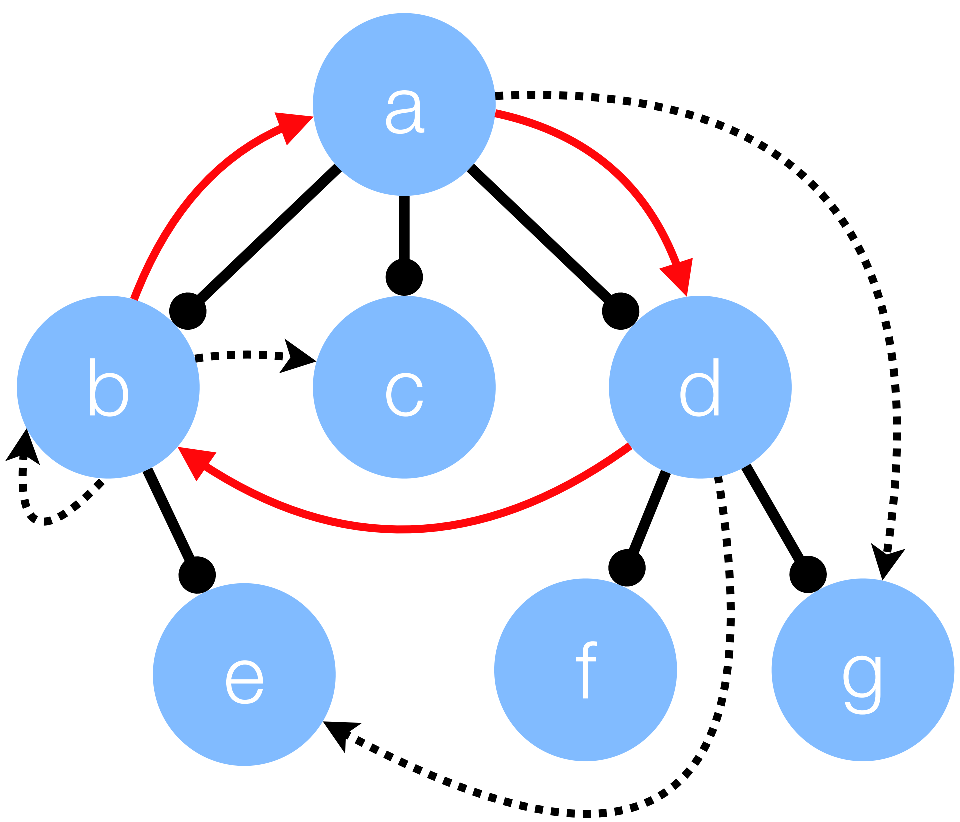
- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow

**Legend:** linear channel, linear process, shared process

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

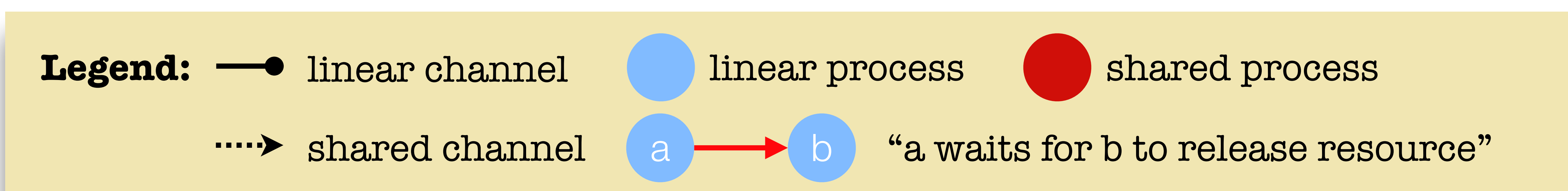
# Let’s add sharing

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



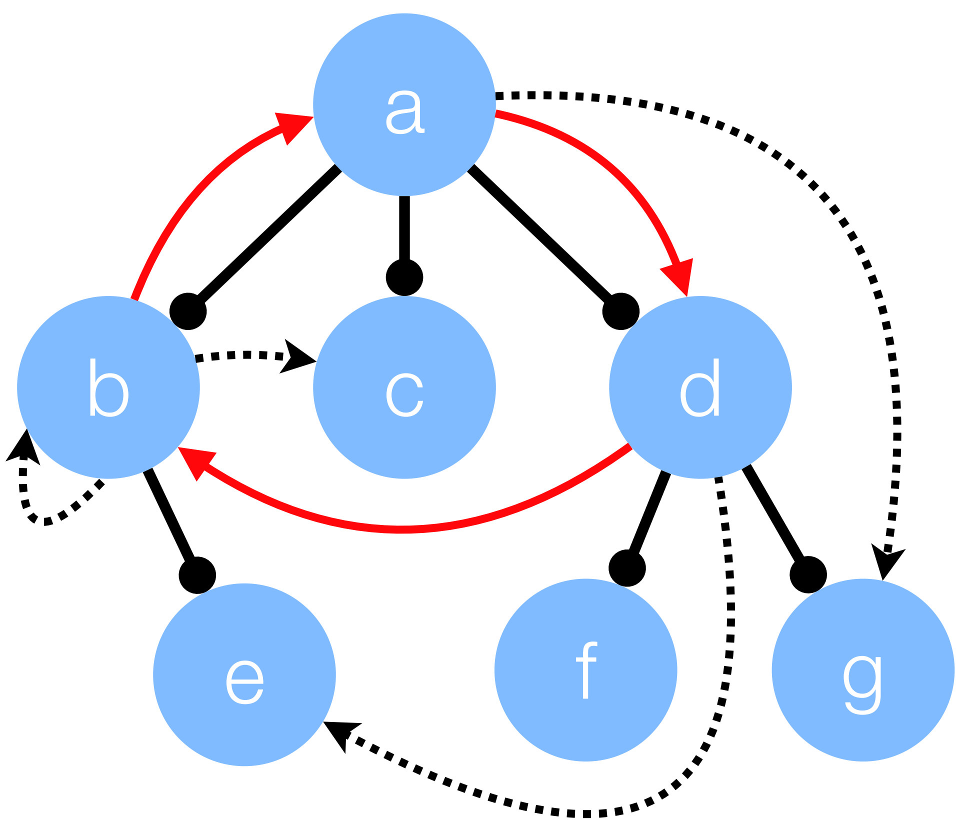

## Acquire-release amounts to “locking”

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



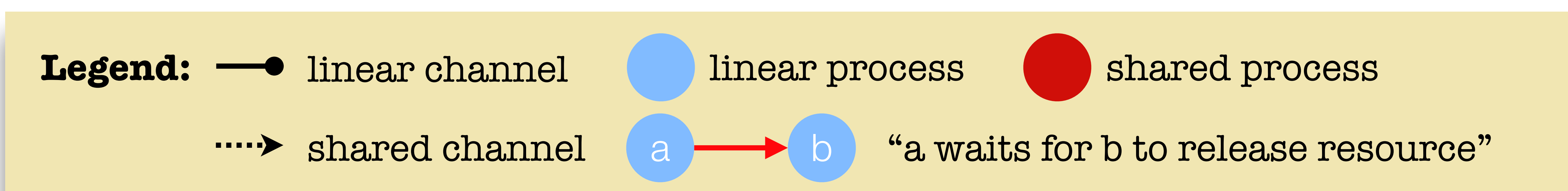
# Let’s add sharing

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




## Acquire-release amounts to “locking”

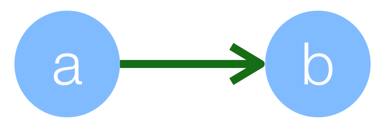
- Possibility of cyclic dependencies
- Let’s visualize this waiting dependency with a red arrow
- Note: red arrows can connect arbitrary nodes in the tree

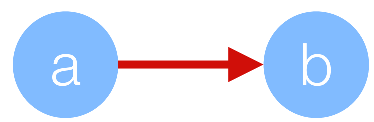


# Two kinds of cycles

# Two kinds of cycles

Two kinds of waiting dependencies:

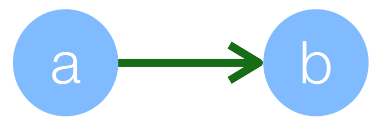
- waiting to synchronize: 

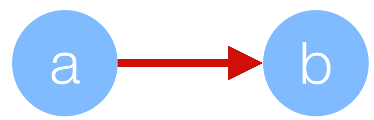
“a waits for b to synchronize”
- waiting to release: 

“a waits for b to release resource”

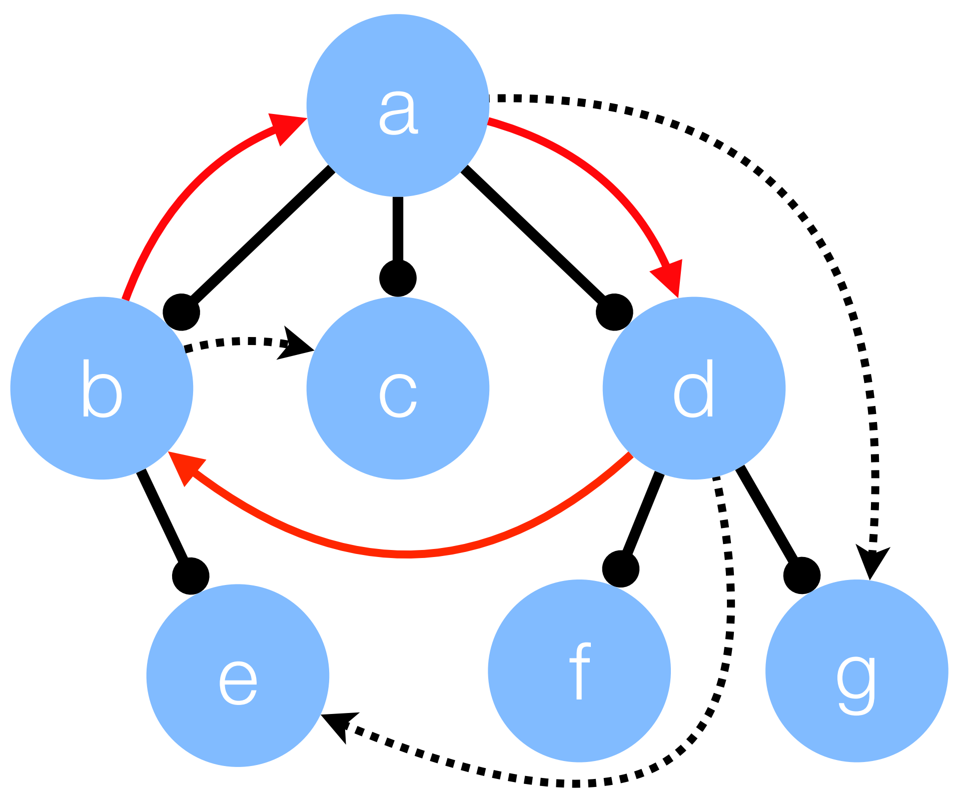
# Two kinds of cycles

Two kinds of waiting dependencies:

- waiting to synchronize: 

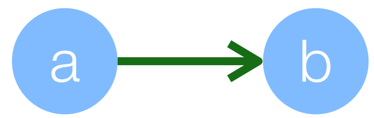
“a waits for b to synchronize”
- waiting to release: 

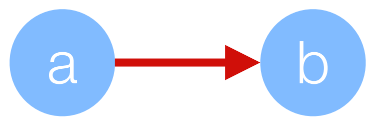
“a waits for b to release resource”



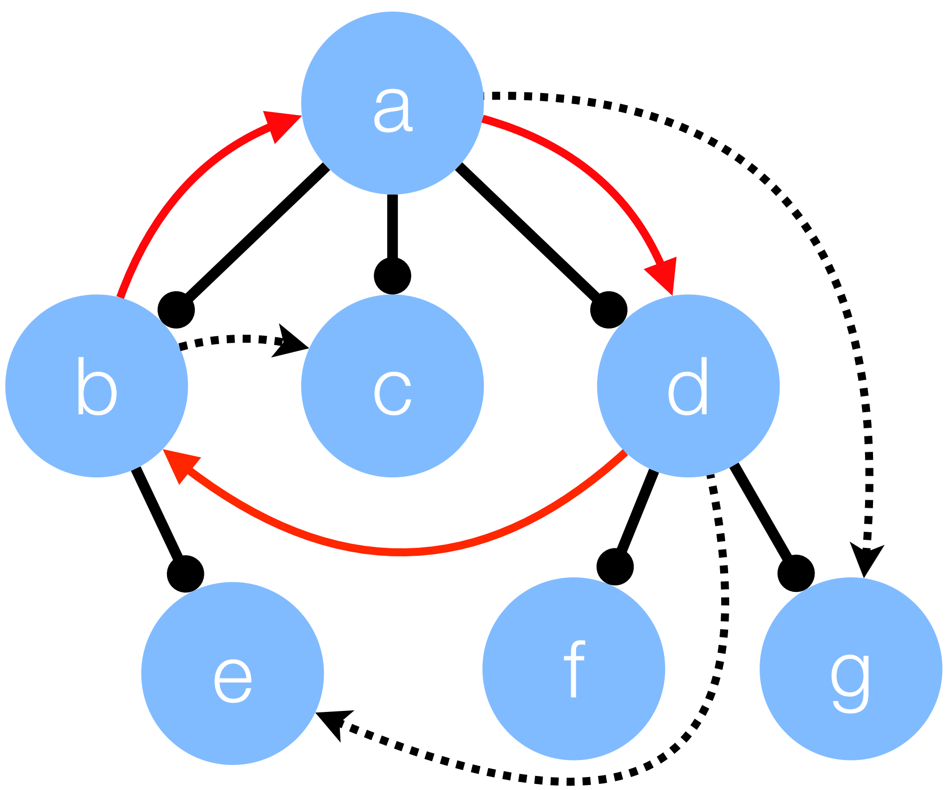
# Two kinds of cycles

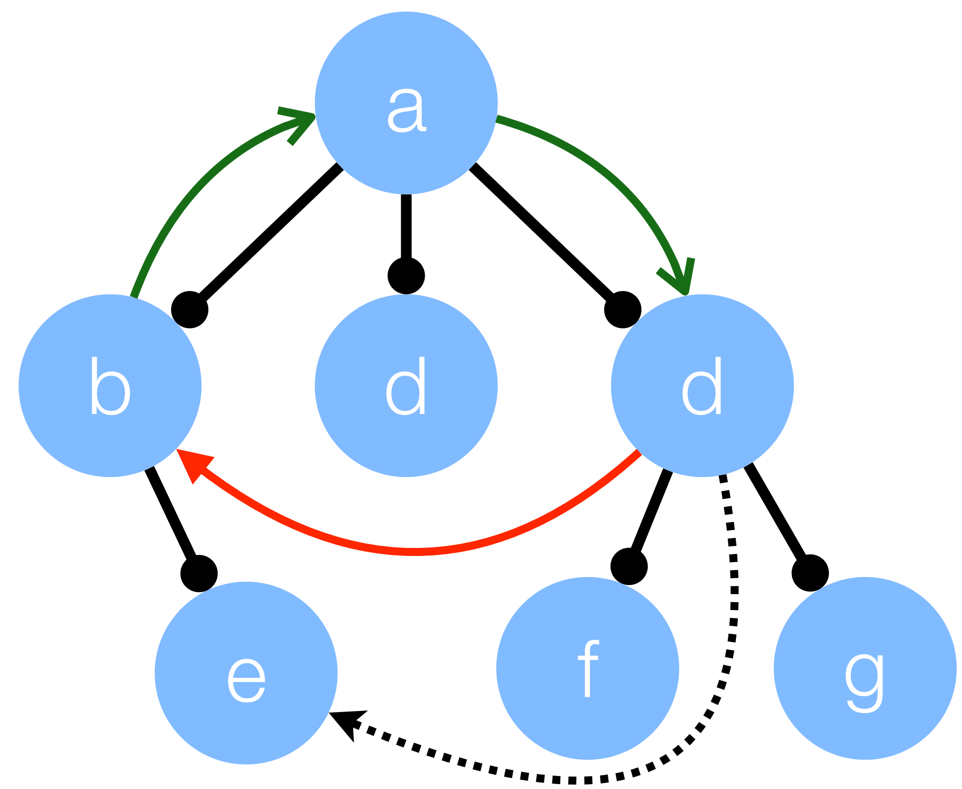
## Two kinds of waiting dependencies:

- waiting to synchronize: 

 "a waits for b to synchronize"
- waiting to release: 

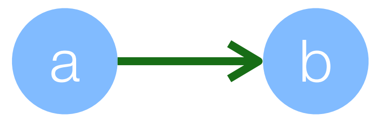
 "a waits for b to release resource"

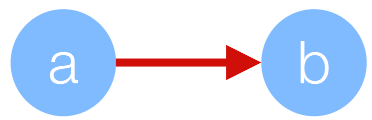


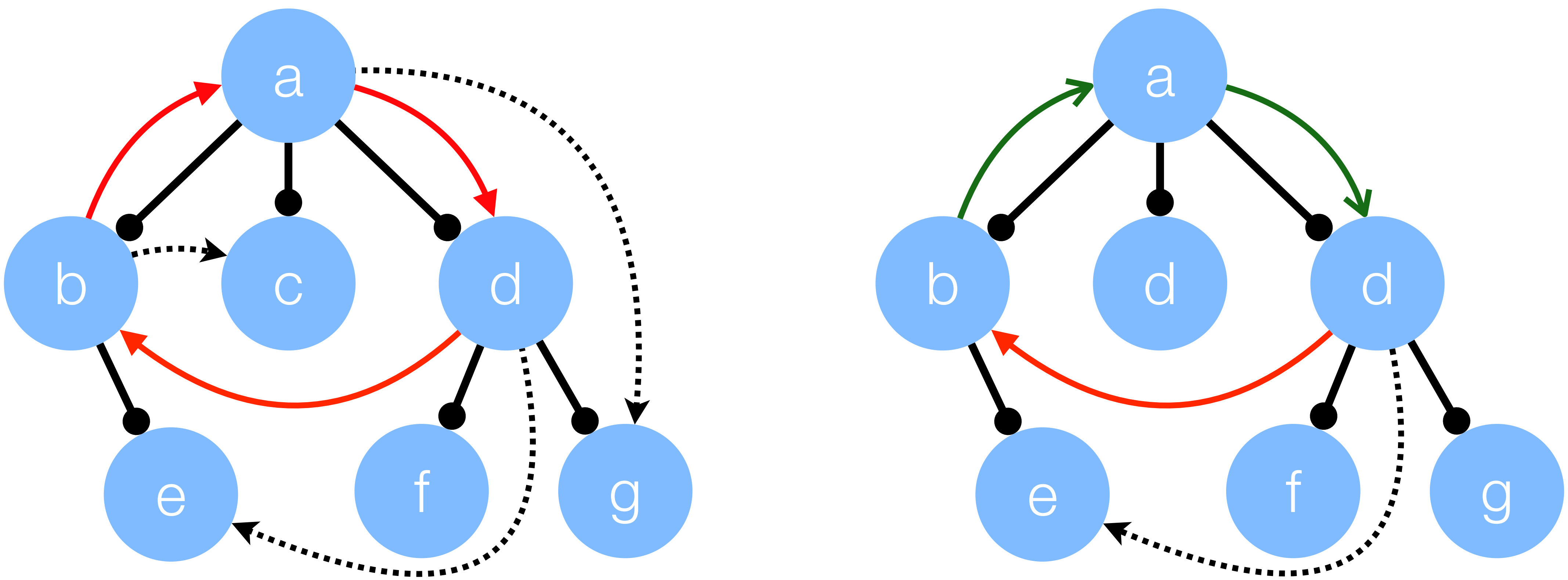
# Two kinds of cycles

## Two kinds of waiting dependencies:

- waiting to synchronize: 

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

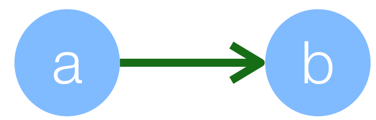
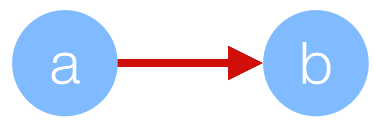
“a waits for b to release resource”



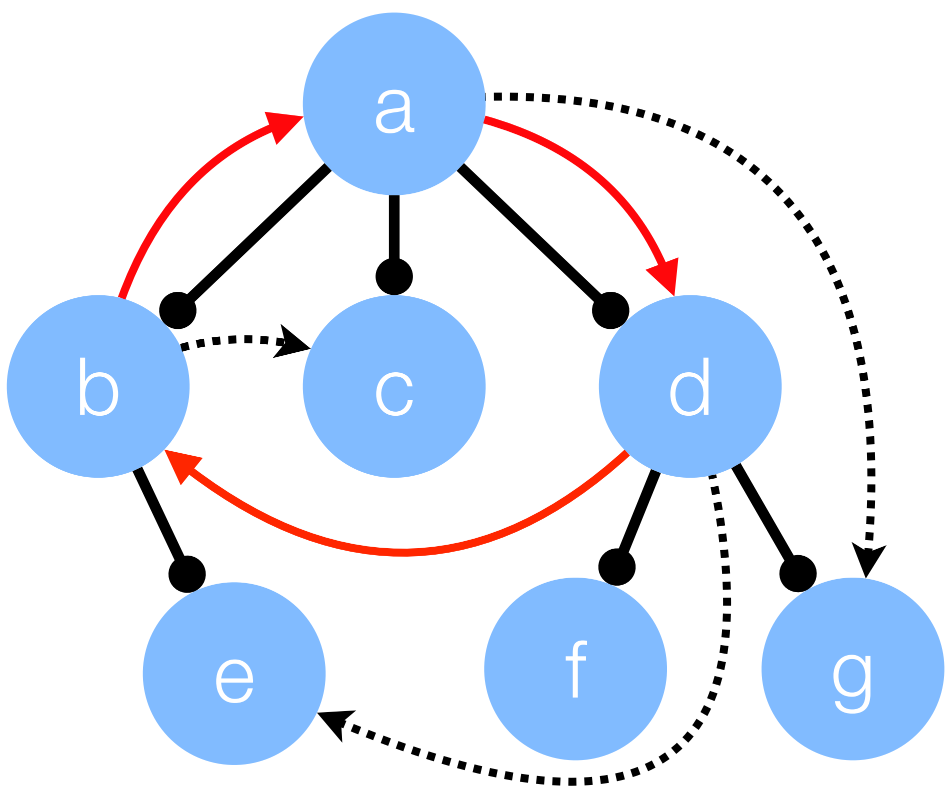

**Cycles can consist of red arrows only or a combination of red and green arrows.**

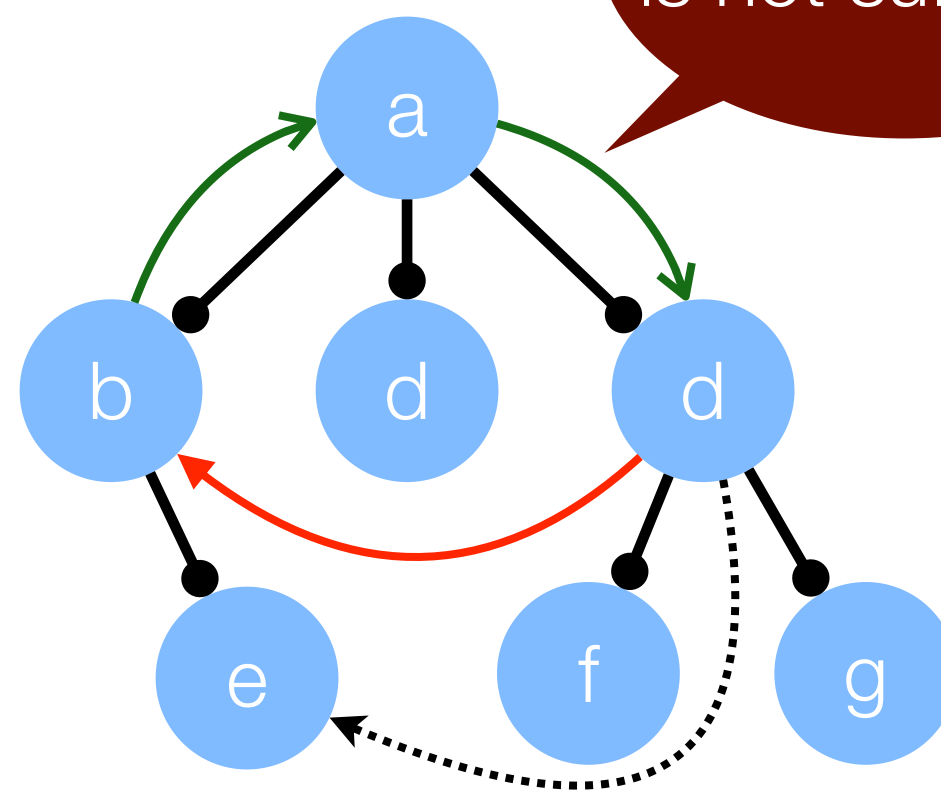
# Two kinds of cycles

## Two kinds of waiting dependencies:

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

“locking-up” is not sufficient!




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

# Idea: competitors and collaborators

# Idea: competitors and collaborators

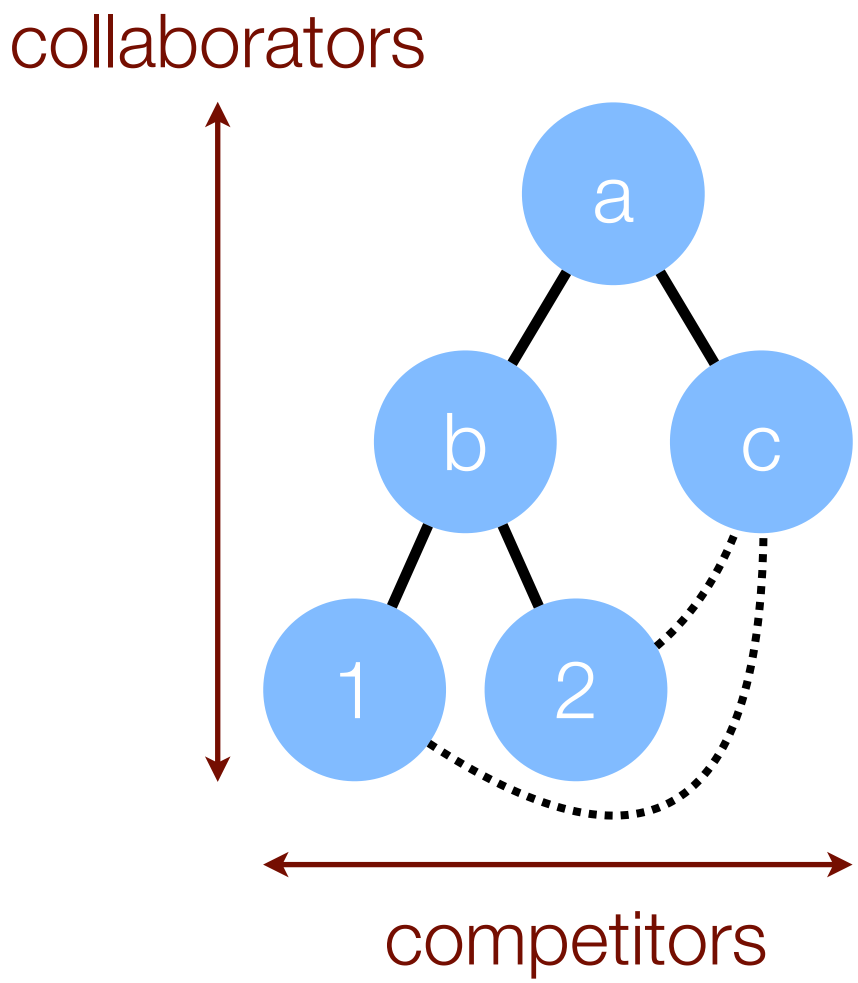
Competitors: overlap in set of resources acquired

# Idea: competitors and collaborators

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

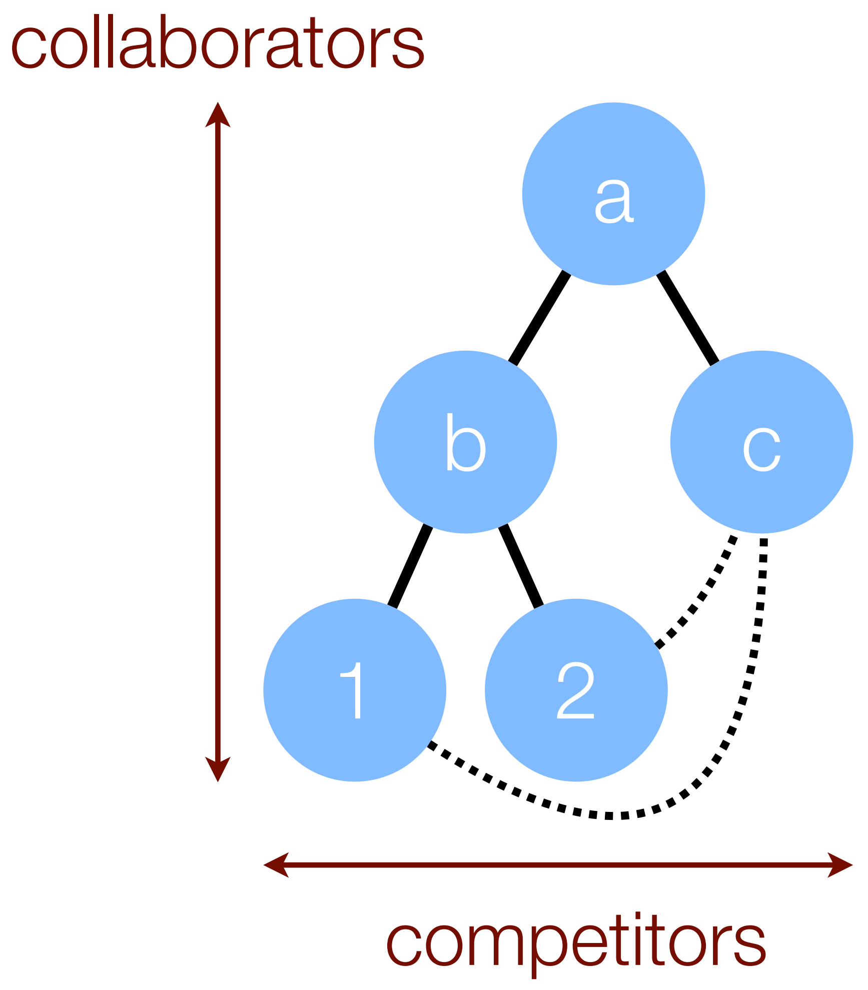
# Idea: competitors and collaborators

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



# Idea: competitors and collaborators

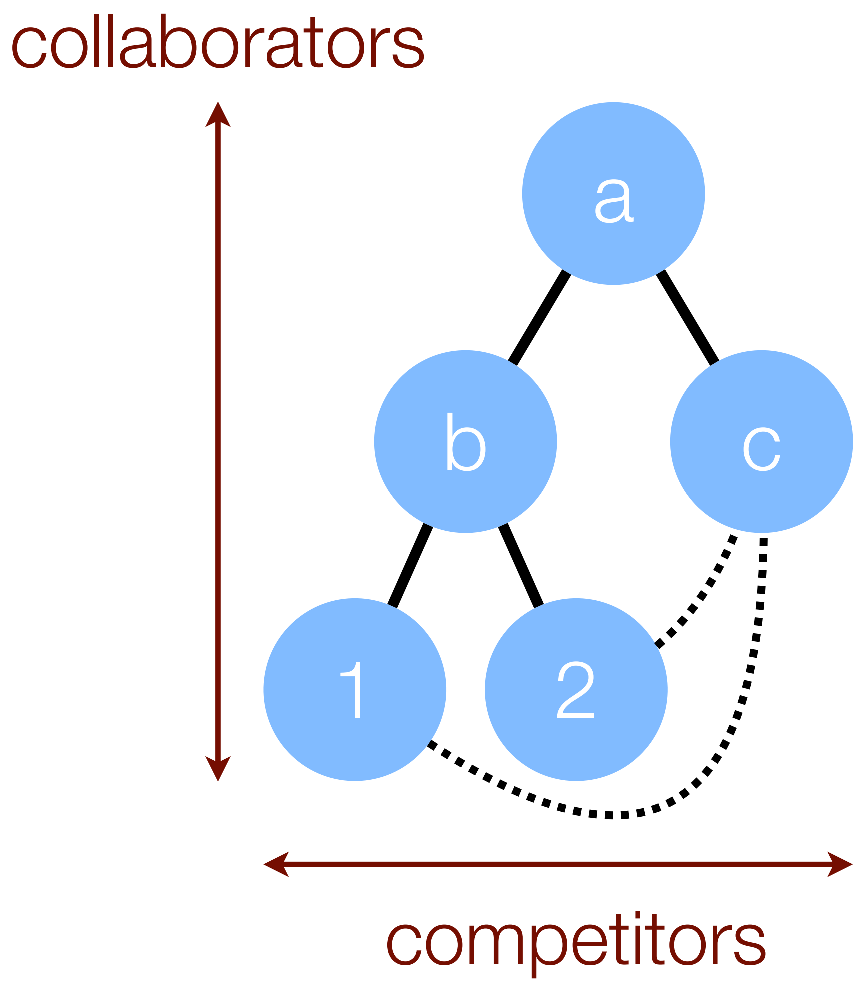
- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired




- competitors tend to be siblings

# Idea: competitors and collaborators

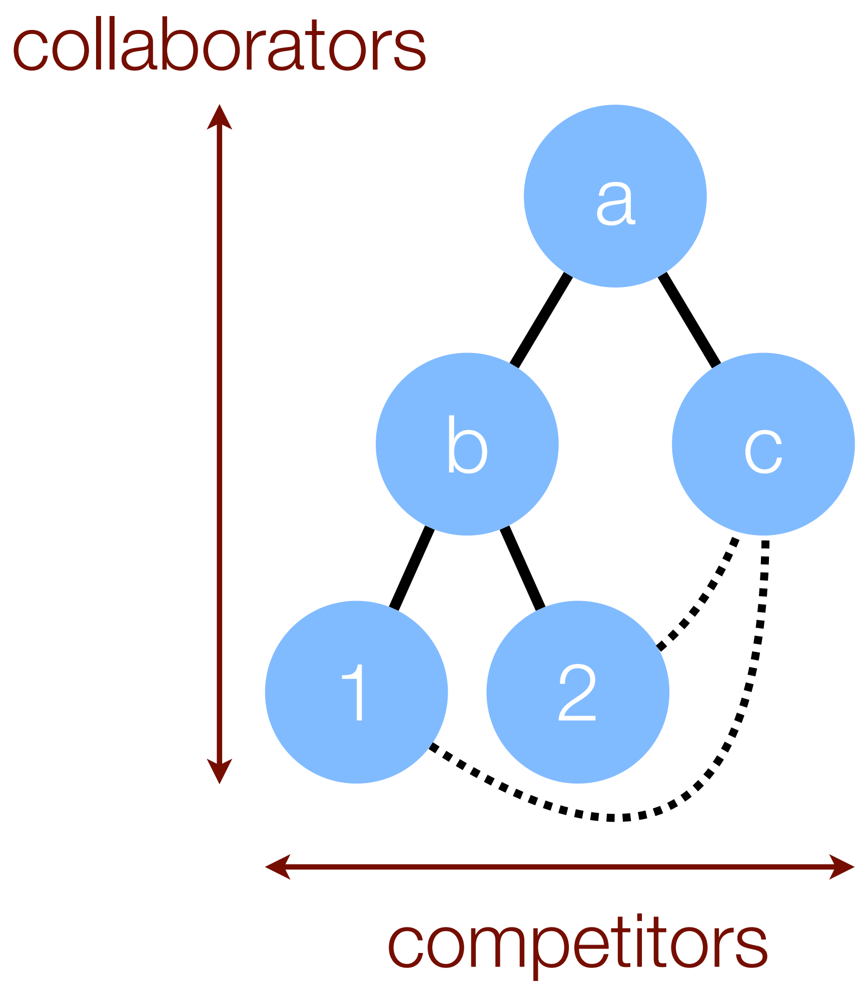
- Competitors: overlap in set of resources acquired
- Collaborators: do not overlap in set of resources acquired




- competitors tend to be siblings
- collaborators tend to be in the same branch

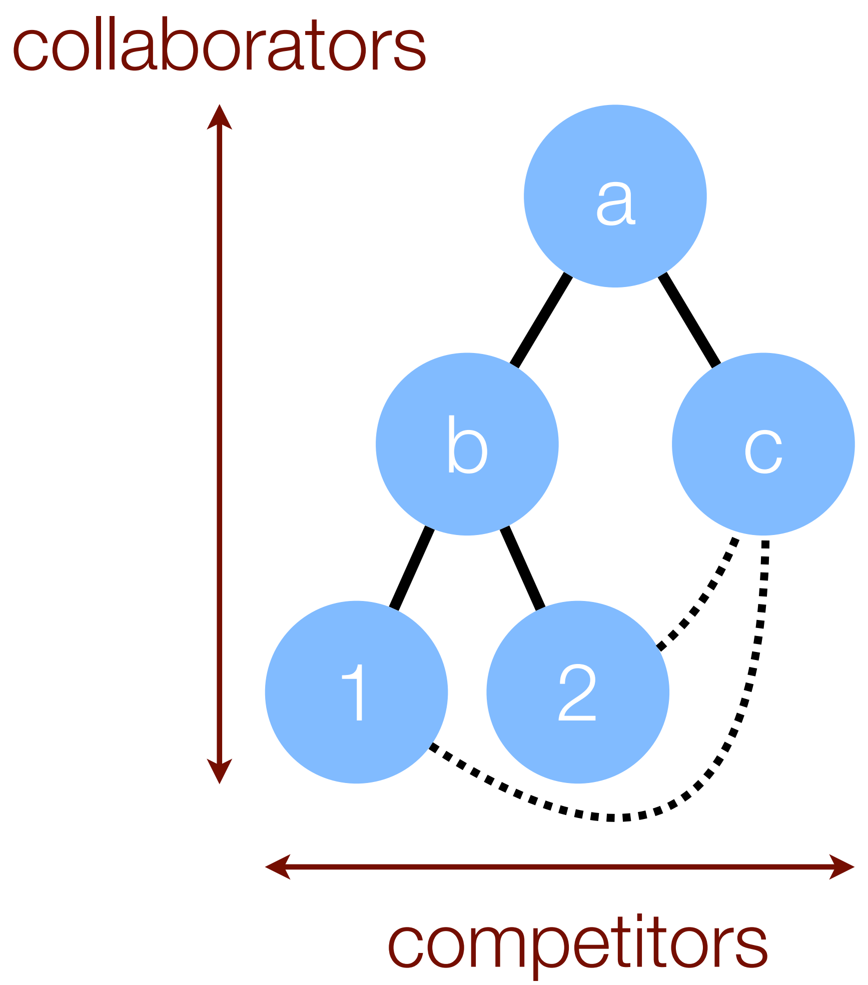
# Manifest deadlock-freedom

**Define type system enforcing the following invariants:**



# Manifest deadlock-freedom

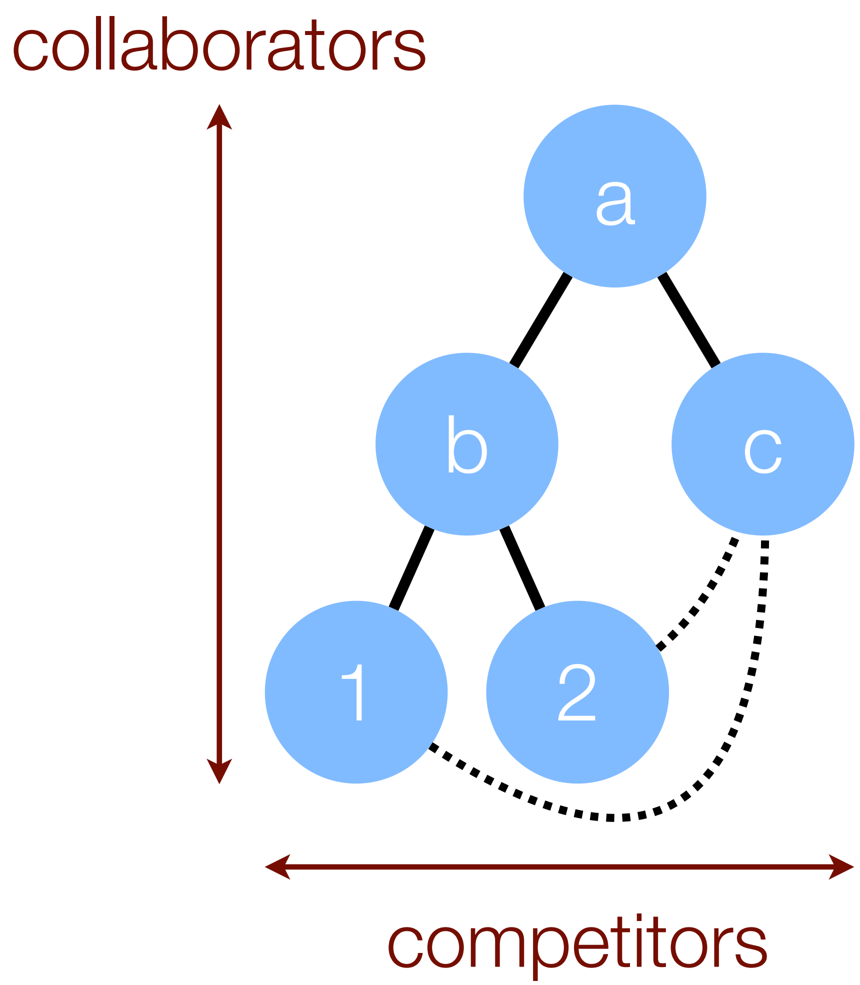
Define type system enforcing the following invariants:




A competitors employ locking-up for resources they compete for

# Manifest deadlock-freedom

**Define type system enforcing the following invariants:**

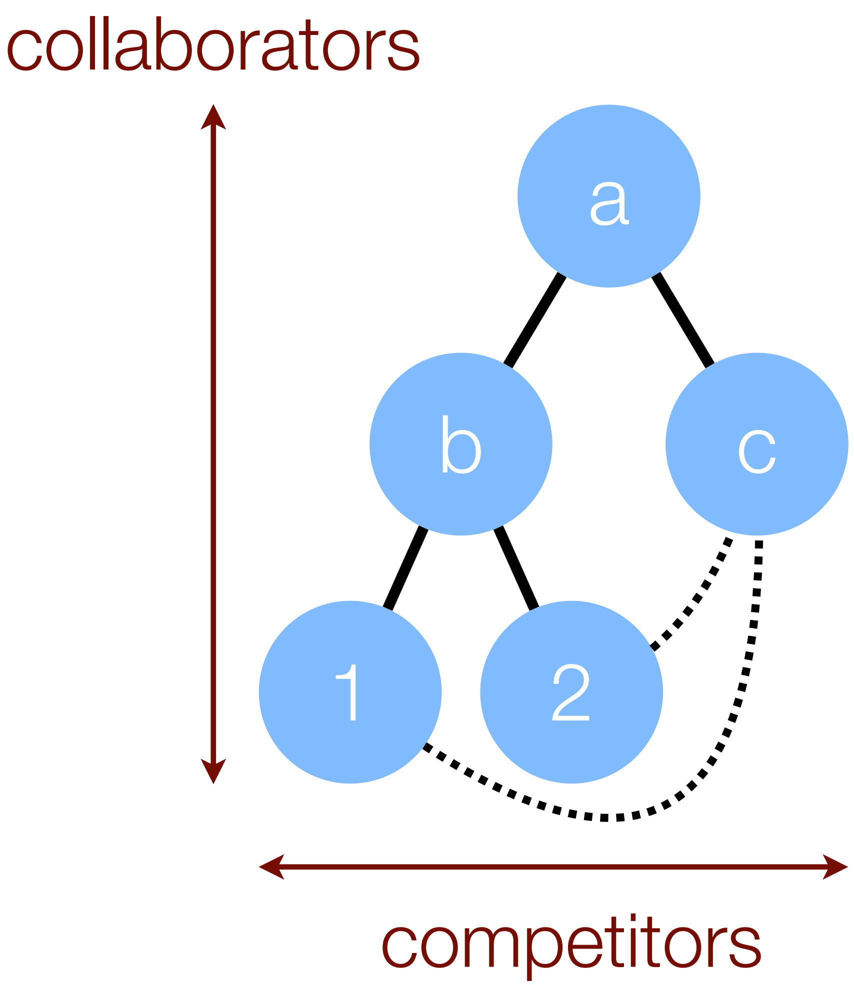



A competitors employ locking-up for resources they compete for

B collaborators acquire mutually disjoint sets of resources

# Manifest deadlock-freedom

Define type system enforcing the following invariants:



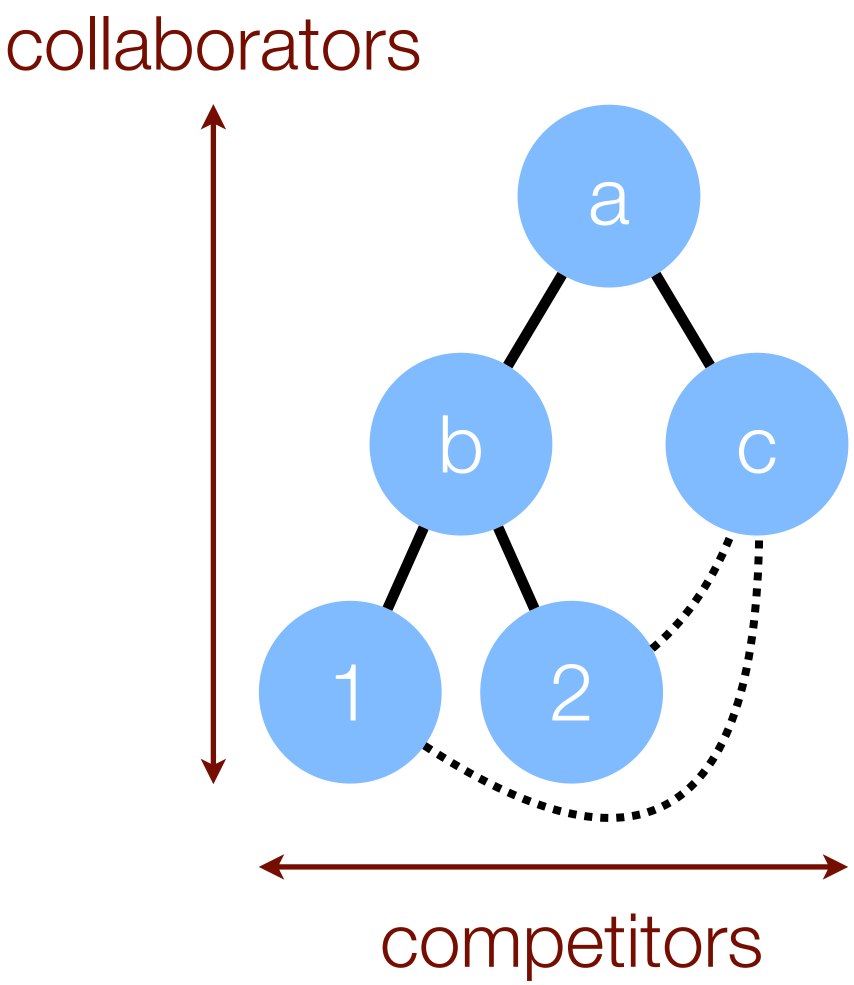

A competitors employ locking-up for resources they compete for

B collaborators acquire mutually disjoint sets of resources

C competitors have released all acquired resources when synchronizing with other competitors (“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 modal worlds, abstract values equipped with a partial order.

# Manifest deadlock-freedom

- Introduce modal worlds, abstract values equipped with a partial order.
  - Every process invariantly resides at a world.

# Manifest deadlock-freedom

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

$$\Psi;\ \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

# Manifest deadlock-freedom

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

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

"Process $P$ provides a session of type $A_m$ along channel $x_m$, using channels in $\Gamma$ (and $\Phi; \Delta$)."

# Manifest deadlock-freedom

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

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

"Process $P$ provides a session of type $A_m$ along channel $x_m$, using channels in $\Gamma$ (and $\Phi; \Delta$)."

Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest Deadlock-Freedom for Shared Session Types. ESOP 2019.

# Manifest deadlock-freedom

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

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

worlds associated
with process

# Manifest deadlock-freedom

Introduce modal worlds, abstract values equipped with a partial order.

Every process invariantly resides at a world.

Every process indicates the range of worlds it may acquire.

$$\Psi;\ \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

# Manifest deadlock-freedom

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

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

self-world:
world at which process
resides

# Manifest deadlock-freedom

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

$$\Psi;\ \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

# Manifest deadlock-freedom

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

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

min-world:
world of minimal resource
to be acquired

# Manifest deadlock-freedom

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

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

# Manifest deadlock-freedom

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

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

max-world: world
of maximal resource to be
acquired

# Manifest deadlock-freedom

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

$$\Psi;\ \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

# Manifest deadlock-freedom

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

$$\boxed{\Psi}; \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\boxed{\Psi}; \Gamma; \Phi; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

partial world order

# Manifest deadlock-freedom

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

# Manifest deadlock-freedom

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

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

# Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

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

# Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

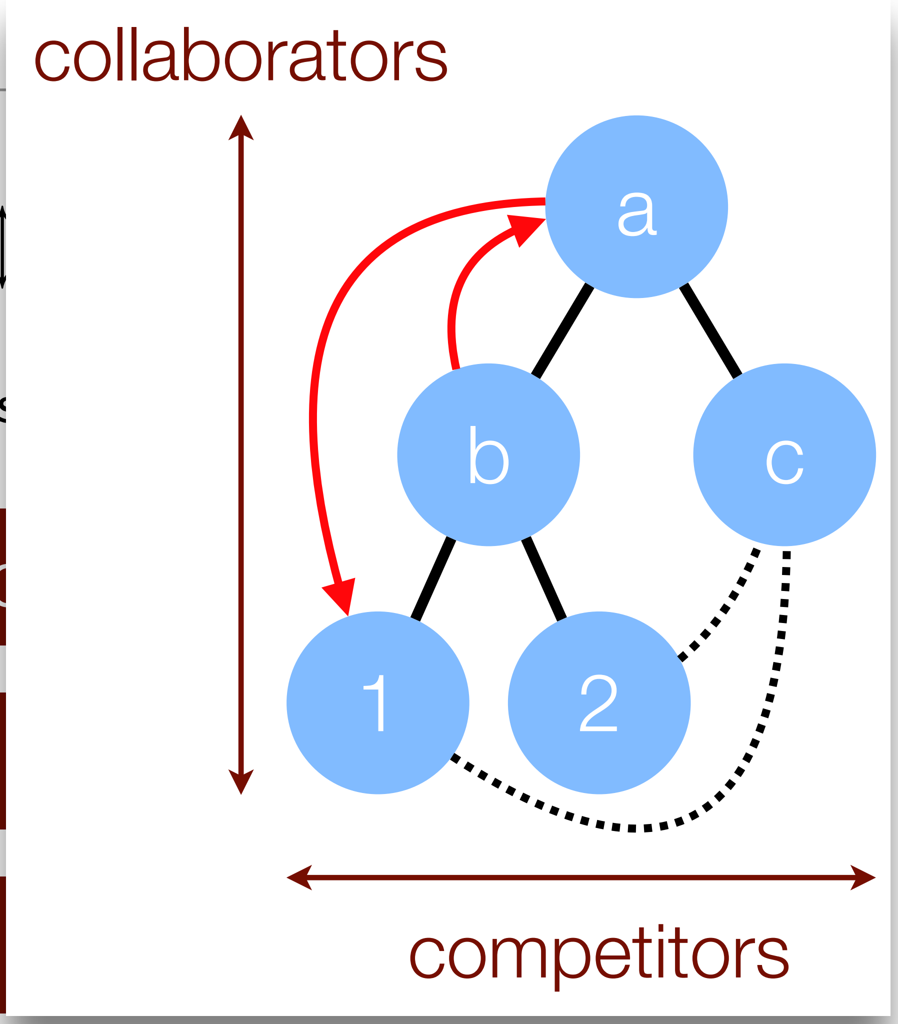
- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)

# Manifest deadlock-freedom

$$\Psi;\ \Gamma \vdash_\Sigma P :: (x_\mathsf{S} : A_\mathsf{S}[\omega_\mathsf{self} \updownarrow$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma P :: (x_\mathsf{L} : A_\mathsf{L}[\omega_\mathsf{s}$$

- 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

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

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

# Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

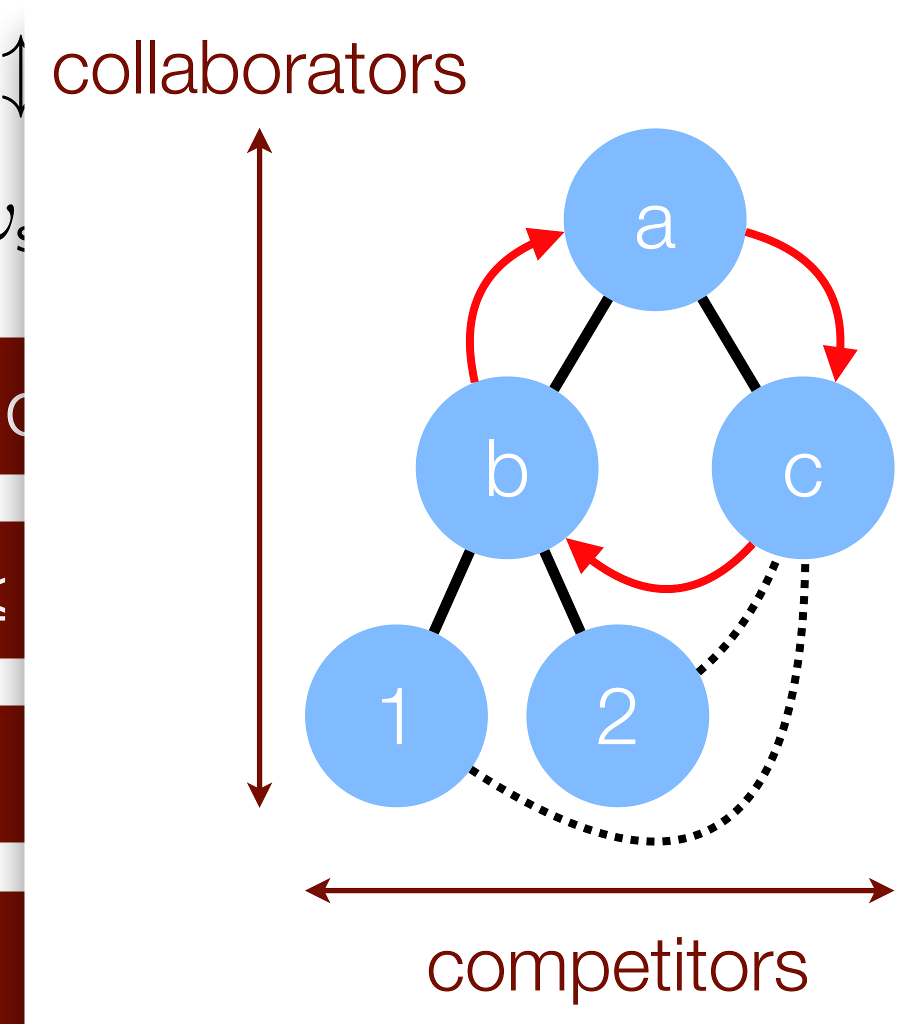
- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up

# Manifest deadlock-freedom

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{s}}$$

- 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

$$\Psi;\, \Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

- 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

$$\Psi;\, \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\, \Gamma;\, \Phi;\, \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

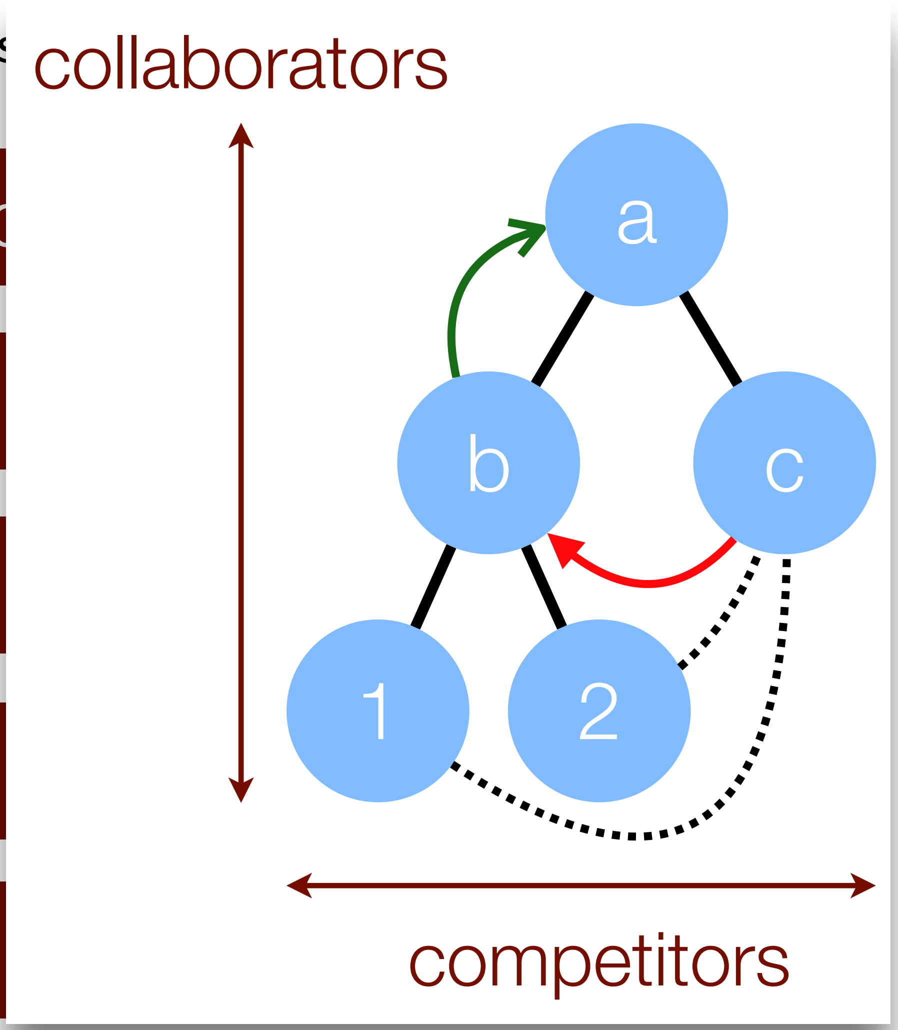
- Express invariants A, B, and C in terms of:
  - min(parent) ≤ self(acquired_child) ≤ max(parent)
  - max(parent) < min(child)
  - for an acquire: lock-up
  - right-rule: all resources released

# Manifest deadlock-freedom

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{s}}$$

- Express invariants A, B, and C in terms o
  - min(parent) ≤ self(acquired_child) ≤
  - max(parent) < min(child)
  - for an acquire: lock-up
  - right-rule: all resources released




**no ingoing red and up-going green arrow**

# Manifest deadlock-freedom

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

- 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
  - right-rule: all resources released

# Manifest deadlock-freedom

$$\Psi;\ \Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

$$\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_{\mathsf{self}} \updownarrow^{\omega_{\mathsf{max}}}_{\omega_{\mathsf{min}}}])$$

- 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
  - right-rule: all resources released
- These low-level invariants are enforced by typing.

# World and order creation

# World and order creation

$$\frac{\Psi, \mathsf{w};\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_{\Sigma} \mathsf{w} \leftarrow \mathsf{new\_world};\ Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}\ (\textsc{T-New}_{\mathsf{L}})$$

# World and order creation

$$\frac{\Psi, \mathsf{w}; \Gamma; \Phi; \Delta \vdash_\Sigma Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma \mathsf{w} \leftarrow \mathsf{new\_world}; Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])} \; (\text{T-N}\textsc{ew}_{\mathsf{L}})$$

# World and order creation

$$\frac{\Psi, \mathsf{w};\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma \mathsf{w} \leftarrow \mathsf{new\_world};\ Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])} \ (\text{T-}\textsc{New}_{\mathsf{L}})$$

$$\frac{\omega_p, \omega_r \in \Psi \qquad (\Psi, \omega_p < \omega_r)^+ \text{ irreflexive} \qquad \Psi, \omega_p < \omega_r;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi;\ \Gamma;\ \Phi;\ \Delta \vdash_\Sigma \omega_p < \omega_r;\ Q :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])} \ (\text{T-}\textsc{Ord}_{\mathsf{L}})$$

# World and order creation

$$\frac{\Psi, \mathsf{w}; \Gamma; \Phi; \Delta \vdash_\Sigma Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma \mathsf{w} \leftarrow \mathsf{new\_world}; Q_{\mathsf{w}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])} \text{ (T-NEW}_{\mathsf{L}}\text{)}$$

$$\frac{\omega_p, \omega_r \in \Psi \qquad (\Psi, \omega_p < \omega_r)^+ \text{ irreflexive} \qquad \Psi, \omega_p < \omega_r; \Gamma; \Phi; \Delta \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi; \Gamma; \Phi; \Delta \vdash_\Sigma \omega_p < \omega_r; Q :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])} \text{ (T-ORD}_{\mathsf{L}}\text{)}$$

# Acquire

# Acquire

$$\frac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\text{L})$$

# Acquire

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\text{L})$$

# Acquire

lock-up

$$\frac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\, \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\, \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\, \Delta \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\, \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\, \Phi;\, \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\; x_{\mathsf{S}}\,; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])} \;(\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

# Acquire

$$\frac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

# Acquire

min(parent) ≤ self(acquired_child) ≤ max(parent)

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}} ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])} \ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

# Acquire

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\, \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\, \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\, \Delta \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\, \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\, \Phi;\, \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\, x_{\mathsf{S}}\, ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\mathrm{T}\text{-}{\uparrow^{\mathsf{S}}_{\mathsf{L}}}\mathrm{L})$$

# Acquire

max(parent) < min(child)

$$\dfrac{\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \qquad \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \qquad \Psi; \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}]; \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}]; \Delta \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}{\Psi; \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}]; \Phi; \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\,; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])} \ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

# Acquire

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\, x_{\mathsf{S}} ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

# Acquire

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}} ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])} \ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

# Acquire

$$\frac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\text{L})$$

$$\frac{\Psi;\ \Gamma;\ \cdot\ ;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi;\ \Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}\ (\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\text{R})$$

# Acquire

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])}\ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

$$\dfrac{\Psi;\ \Gamma;\ \cdot\ ;\ \cdot \vdash_\Sigma P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi;\ \Gamma \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}\ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{R})$$

# Acquire

$$\dfrac{\begin{array}{c}\forall y_{\mathsf{L}} : B_{\mathsf{L}}[\omega_l \updownarrow^{\omega_r}_{\omega_p}] \in \Phi : \omega_l < \omega_m \\ \Psi^* \vdash \omega_k \leq \omega_m \leq \omega_n \qquad \Psi^+ \vdash \omega_n < \omega_u \\ \Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi, x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])\end{array}}{\Psi;\ \Gamma, x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}];\ \Phi;\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}}[\omega_j \updownarrow^{\omega_n}_{\omega_k}])} \ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

$$\dfrac{\Psi;\ \Gamma;\ \cdot\ ;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])}{\Psi;\ \Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \boxed{\mathsf{accept}\ x_{\mathsf{S}}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}[\omega_m \updownarrow^{\omega_v}_{\omega_u}])} \ (\mathrm{T}\text{-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{R})$$

Φ must be empty

# Remaining typing rules

## Largely unchanged, except for

- right-rules, which must have an empty Φ
- spawn, which must establish invariants

## Compositionality

- Process definitions specify a process' order
- Order of spawner must entail order of spawnee

# Catching potential deadlock

# Catching potential deadlock

$$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{array}{l}
o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ; \\
\qquad\qquad\qquad sr \leftarrow \mathsf{release}\ lr \ ; \mathsf{close}\ o
\end{array}$$

$$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{array}{l}
c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad c.\mathsf{ping} \ ; \\
\quad sr \leftarrow \mathsf{release}\ lr \ ; \mathsf{close}\ c
\end{array}$$

# Catching potential deadlock

$$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$
\begin{array}{l}
o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \mathsf{release}\ lr \ ; \mathsf{close}\ o
\end{array}
$$

$$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$
\begin{array}{l}
c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad lr \leftarrow \mathsf{acquire}\ sr \ ; \\
\quad c.\mathsf{ping} \ ; \\
\quad sr \leftarrow \mathsf{release}\ lr \ ; \mathsf{close}\ c
\end{array}
$$

# Catching potential deadlock

$$
\begin{array}{l}
\mathit{owner} : \{\boxed{\delta_0 < \delta_1 < \delta_2} \vdash \mathbf{1}[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \textsf{sres}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}]\} \\
o[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathit{owner} \leftarrow \mathit{sr}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\textsf{ping} : \mathbf{1}\}[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr} \ ; \\
\quad \mathit{lr} \leftarrow \textsf{acquire}\ \mathit{sr} \ ; \\
\quad \textsf{case}\ c\ \textsf{of} \\
\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \ ; \\
\qquad\qquad\quad \mathit{sr} \leftarrow \textsf{release}\ \mathit{lr} \ ; \textsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
\mathit{contester} : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\textsf{ping} : \mathbf{1}\}[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \textsf{sres}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}]\} \\
c[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}] = \\
\quad \mathit{lr} \leftarrow \textsf{acquire}\ \mathit{sr} \ ; \\
\quad c.\textsf{ping} \ ; \\
\quad \mathit{sr} \leftarrow \textsf{release}\ \mathit{lr} \ ; \textsf{close}\ c
\end{array}
$$

# Catching potential deadlock

$$
\begin{array}{l}
\mathit{owner} : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}]\} \\
o[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathit{owner} \leftarrow \mathit{sr}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr} \; ; \\
\quad \mathit{lr} \leftarrow \mathsf{acquire}\ \mathit{sr} \; ; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \; ; \\
\qquad\qquad\quad \mathit{sr} \leftarrow \mathsf{release}\ \mathit{lr} \; ; \mathsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
\mathit{contester} : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}]\} \\
c[\delta_0 {\updownarrow}^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr}[\delta_1 {\updownarrow}^{\delta_2}_{\delta_2}] = \\
\quad \mathit{lr} \leftarrow \mathsf{acquire}\ \mathit{sr} \; ; \\
\quad c.\mathsf{ping} \; ; \\
\quad \mathit{sr} \leftarrow \mathsf{release}\ \mathit{lr} \; ; \mathsf{close}\ c
\end{array}
$$

# Catching potential deadlock

$$\mathit{owner} : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{array}{l}
o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathit{owner} \leftarrow \mathit{sr}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr} \ ; \\
\quad \mathit{lr} \leftarrow \mathsf{acquire}\ \mathit{sr} \ ; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ; \\
\qquad\qquad\quad \mathit{sr} \leftarrow \mathsf{release}\ \mathit{lr} \ ; \mathsf{close}\ o
\end{array}$$

$$\mathit{contester} : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{array}{l}
c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad \mathit{lr} \leftarrow \mathsf{acquire}\ \mathit{sr} \ ; \\
\quad c.\mathsf{ping} \ ; \\
\quad \mathit{sr} \leftarrow \mathsf{release}\ \mathit{lr} \ ; \mathsf{close}\ c
\end{array}$$

# Catching potential deadlock

$$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \textsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{array}{l}
o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\textsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \ ; \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad \textsf{case}\ c\ \textsf{of} \\
\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \ ; \\
\qquad\qquad\quad sr \leftarrow \textsf{release}\ lr \ ; \textsf{close}\ o
\end{array}$$

$$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\textsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \textsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{array}{l}
c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad lr \leftarrow \textsf{acquire}\ sr \ ; \\
\quad c.\textsf{ping} \ ; \\
\quad sr \leftarrow \textsf{release}\ lr \ ; \textsf{close}\ c
\end{array}$$

# Catching potential deadlock

$$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$$

$$\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \ ;$$

$$\quad lr \leftarrow \mathsf{acquire}\ sr \ ;$$

$$\quad \mathsf{case}\ c\ \mathsf{of}$$

$$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ;$$

$$\qquad\qquad sr \leftarrow \mathsf{release}\ lr \ ; \mathsf{close}\ o$$

$$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$$

$$\quad lr \leftarrow \mathsf{acquire}\ sr \ ;$$

$$\quad c.\mathsf{ping} \ ;$$

$$\quad sr \leftarrow \mathsf{release}\ lr \ ; \mathsf{close}\ c$$

# Catching potential deadlock

$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$

$o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$

$\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr\ ;$

$\quad lr \leftarrow \mathsf{acquire}\ sr\ ;$

$\quad \mathsf{case}\ c\ \mathsf{of}$

$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c\ ;$

$\qquad\qquad sr \leftarrow \mathsf{release}\ lr\ ;\ \mathsf{close}\ o$

$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$

$c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$

$\quad lr \leftarrow \mathsf{acquire}\ sr\ ;$

$\quad c.\mathsf{ping}\ ;$

$\quad sr \leftarrow \mathsf{release}\ lr\ ;\ \mathsf{close}\ c$

# Catching potential deadlock

$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$

$o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$

$\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr\ ;$

$\quad lr \leftarrow \mathsf{acquire}\ sr\ ;$

$\quad \mathsf{case}\ c\ \mathsf{of}$

$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c\ ;$

$\qquad\qquad sr \leftarrow \mathsf{release}\ lr\ ;\ \mathsf{close}\ o$

$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$

$c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$

$\quad lr \leftarrow \mathsf{acquire}\ sr\ ;$

$\quad c.\mathsf{ping}\ ;$

$\quad sr \leftarrow \mathsf{release}\ lr\ ;\ \mathsf{close}\ c$

# Catching potential deadlock

$$
\begin{array}{l}
\mathit{owner} : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\} \\
o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathit{owner} \leftarrow \mathit{sr}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr} \;; \\
\quad \mathit{lr} \leftarrow \mathsf{acquire}\ \mathit{sr} \;; \\
\quad \mathsf{case}\ c\ \mathsf{of} \\
\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \;; \\
\qquad\qquad\qquad \mathit{sr} \leftarrow \mathsf{release}\ \mathit{lr} \;; \mathsf{close}\ o
\end{array}
$$

$$
\begin{array}{l}
\mathit{contester} : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\} \\
c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathit{contester} \leftarrow \mathit{sr}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
\quad \mathit{lr} \leftarrow \mathsf{acquire}\ \mathit{sr} \;; \\
\quad c.\mathsf{ping} \;; \\
\quad \mathit{sr} \leftarrow \mathsf{release}\ \mathit{lr} \;; \mathsf{close}\ c
\end{array}
$$

# Catching potential deadlock

$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$

$o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$

$\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \ ;$

$\quad lr \leftarrow \mathsf{acquire}\ sr \ ;$

$\quad \mathsf{case}\ c\ \mathsf{of}$

$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \ ;$

$\qquad\qquad sr \leftarrow \mathsf{release}\ lr \ ; \ \mathsf{close}\ o$

$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$

$c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$

$\quad lr \leftarrow \mathsf{acquire}\ sr \ ;$

$\quad c.\mathsf{ping} \ ;$

$\quad sr \leftarrow \mathsf{release}\ lr \ ; \ \mathsf{close}\ c$

# Catching potential deadlock

$$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \textsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{aligned}
&o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
&\quad c : \oplus\{\textsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \;; \\
&\quad lr \leftarrow \textsf{acquire}\ sr \;; \\
&\quad \textsf{case}\ c\ \textsf{of} \\
&\quad \mid \textsf{ping} \rightarrow \textsf{wait}\ c \;; \\
&\qquad\qquad\quad sr \leftarrow \textsf{release}\ lr \;;\ \textsf{close}\ o
\end{aligned}$$

not a collaborator!

$$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\textsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \textsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$\begin{aligned}
&c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] = \\
&\quad lr \leftarrow \textsf{acquire}\ sr \;; \\
&\quad c.\textsf{ping} \;; \\
&\quad sr \leftarrow \textsf{release}\ lr \;;\ \textsf{close}\ c
\end{aligned}$$

# Catching potential deadlock

$$owner : \{\delta_0 < \delta_1 < \delta_2 \vdash \mathbf{1}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$o[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow owner \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$$

$\quad c : \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr \;;$

$\quad lr \leftarrow \mathsf{acquire}\ sr \;;$

$\quad \mathsf{case}\ c\ \mathsf{of}$

$\quad \mid \mathsf{ping} \rightarrow \mathsf{wait}\ c \;;$

$\qquad\qquad sr \leftarrow \mathsf{release}\ lr \;; \mathsf{close}\ o$

not a collaborator!

$$contester : \{\delta_0 < \delta_1 < \delta_2 \vdash \oplus\{\mathsf{ping} : \mathbf{1}\}[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow \mathsf{sres}[\delta_1 \updownarrow^{\delta_2}_{\delta_2}]\}$$

$$c[\delta_0 \updownarrow^{\delta_1}_{\delta_1}] \leftarrow contester \leftarrow sr[\delta_1 \updownarrow^{\delta_2}_{\delta_2}] =$$

$\quad lr \leftarrow \mathsf{acquire}\ sr \;;$

$\quad c.\mathsf{ping} \;;$

$\quad sr \leftarrow \mathsf{release}\ lr \;; \mathsf{close}\ c$

not all resources released!

# Taking stock

# Taking stock

- Type systems allows us to write interesting and common programs that are guaranteed to be deadlock-free, e.g.,:
  - dining philosophers (see paper)
  - imperative queue (see paper)

# Taking stock

- Type systems allows us to write interesting and common programs that are guaranteed to be deadlock-free, e.g.,:
  - dining philosophers (see paper)
  - imperative queue (see paper)
- Currently not supported:
  - unbounded circular process networks
  - world inference

The end — or the beginning for you?

# Related work (not a comprehensive list!)

## Session types and multiparty session types:


- Kohei Honda. Types for Dyadic Interaction. CONCUR 1993.
- Kohei Honda, Vasco Thudichum Vasconcelos, Makoto Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. ESOP 1998.
- Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida, Sophia Drossopoulou. Session Types for Object-Oriented Languages. ECOOP 2006. Kohei Honda, Nobuko Yoshida, Marco Carbone. Multiparty asynchronous session types. POPL 2008.
- Simon J. Gay, Malcolm Hole. Subtyping for session types in the pi calculus. Acta Informatica 42(2-3): 191-225 (2005).

# Related work (not a comprehensive list!)

## Intuitionistic linear logic session types:


- Luís Caires, Frank Pfenning. Session Types as Intuitionistic Linear Propositions. CONCUR 2010.
- Bernardo Toninho, Luís Caires, Frank Pfenning. Higher-Order Processes, Functions, and Sessions: A Monadic Integration. ESOP 2013.
- Luís Caires, Jorge A. Pérez, Frank Pfenning, Bernardo Toninho. Behavioral Polymorphism and Parametricity in Session-Based Communication. ESOP 2013.

# Related work (not a comprehensive list!)

## Classical linear logic session types:

- Philip Wadler. Propositions as sessions. ICFP 2012.
- Sam Lindley, J. Garrett Morris. A Semantics for Propositions as Sessions. ESOP 2015.
- Sam Lindley, J. Garrett Morris. Talking bananas: structural recursion for session types. ICFP 2016.
- Atsushi Igarashi, Peter Thiemann, Vasco T. Vasconcelos, Philip Wadler. Gradual session types. ICFP 2017.
- Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. To appear at ICFP 2021.
- Pedro Rocha and Luis Caires. Propositions-as-types and shared state. To appear at ICFP 2021.

# Related work (not a comprehensive list!)

## Various:

- Ankush Das, Jan Hoffmann, Frank Pfenning. Work Analysis with Resource-Aware Session Types. LICS 2018.
- Resource-aware session types for digital contracts. Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. CSF 2021.
- Farzaneh Derakhshan, Stephanie Balzer, and Limin Jia. Session Logical Relations for Noninterference. LICS 2021.

→ how to find papers?

→ dblp (https://dblp.uni-trier.de/)

→ Google Scholar (https://scholar.google.com/)