Capabilities, Session Types and Active Objects

Marco Patrignani [1] Dave Clarke [1,2]

[1]iMinds-DistriNet, Dept. Computer Science, KU Leuven

[2]CS Division, Dept. Information Technology, Uppsala University

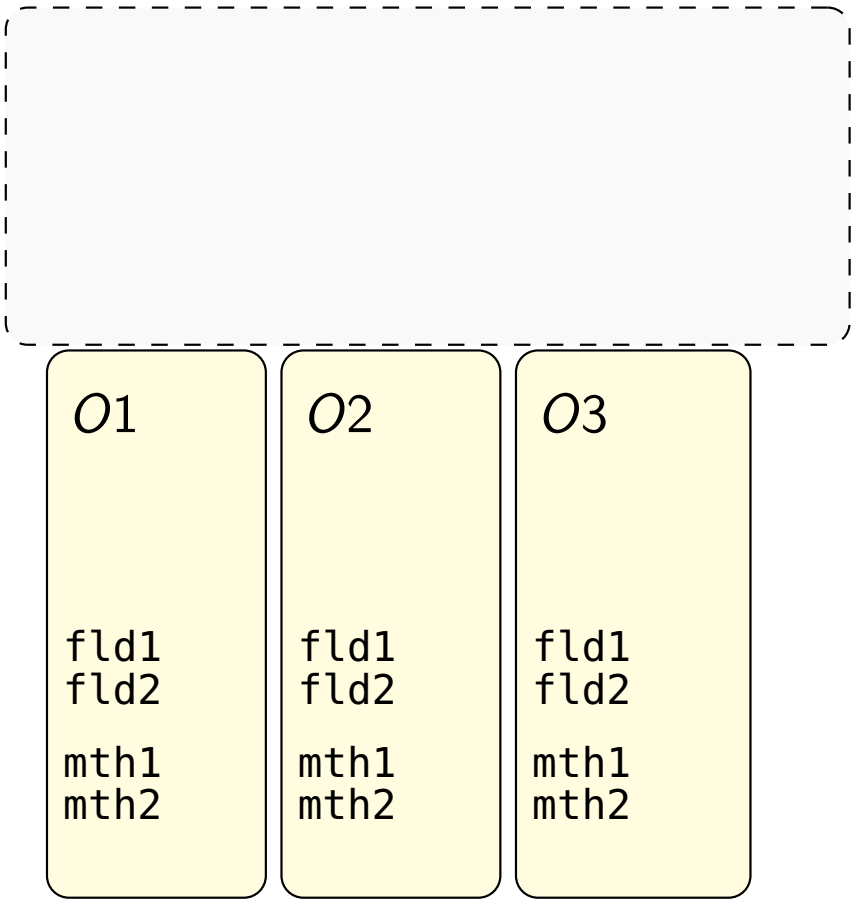
January 14, 2014

Outline

Overview

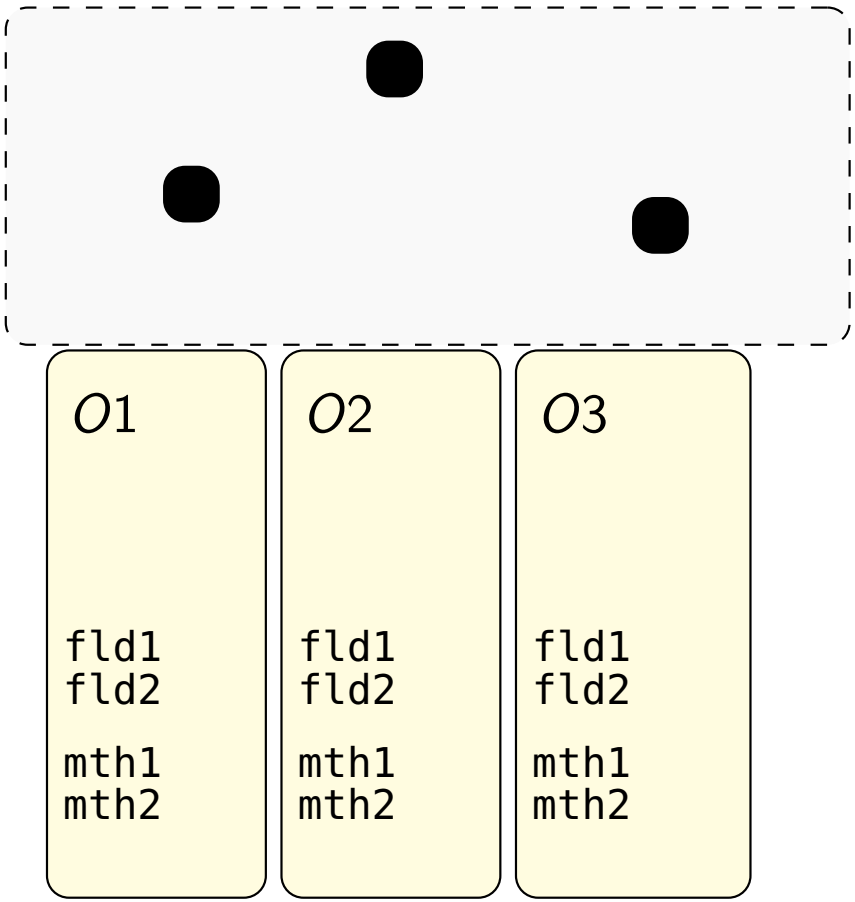
- global heap

Overview



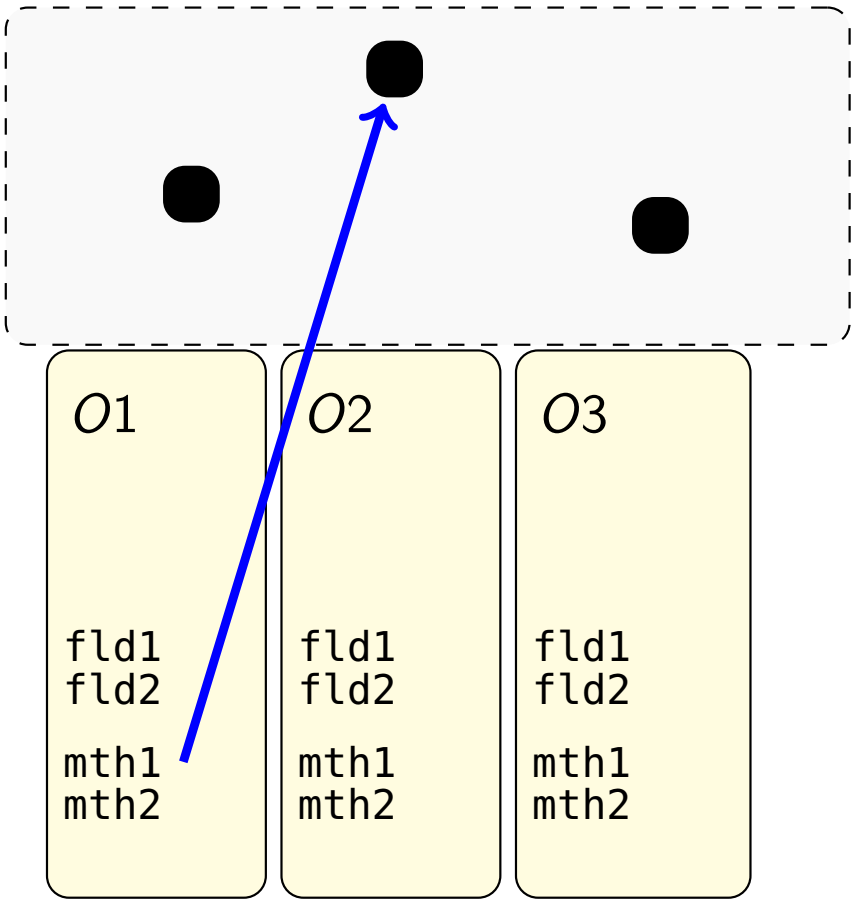
- global heap
- active objects

Overview



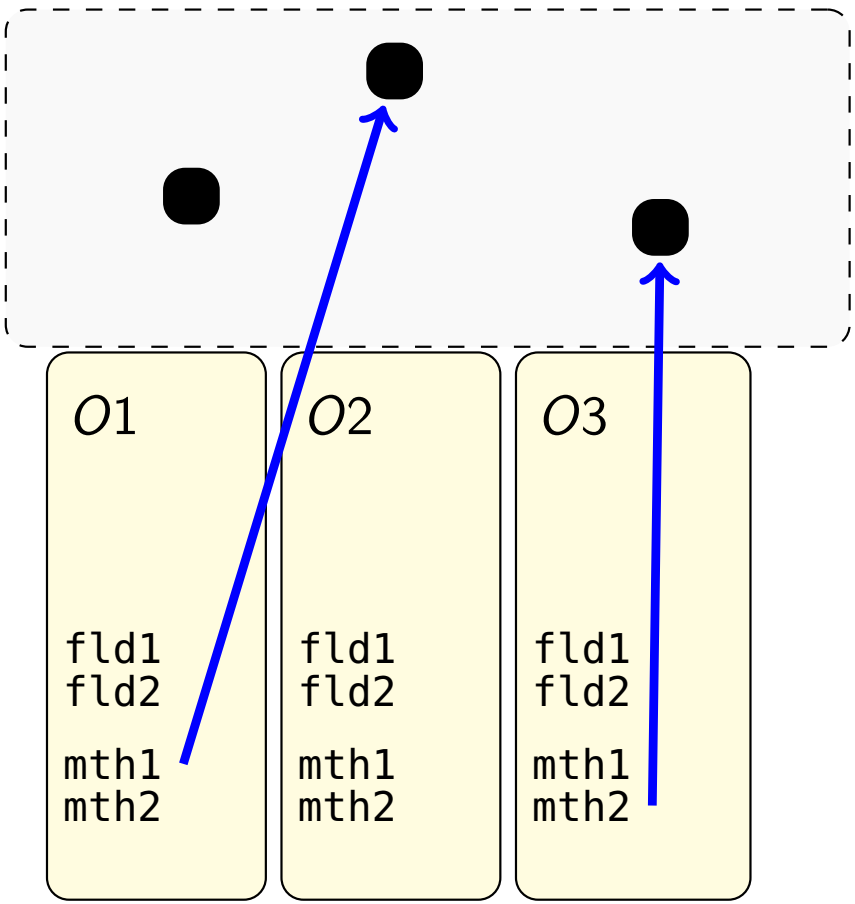
- global heap
- active objects
- shared resources

Overview



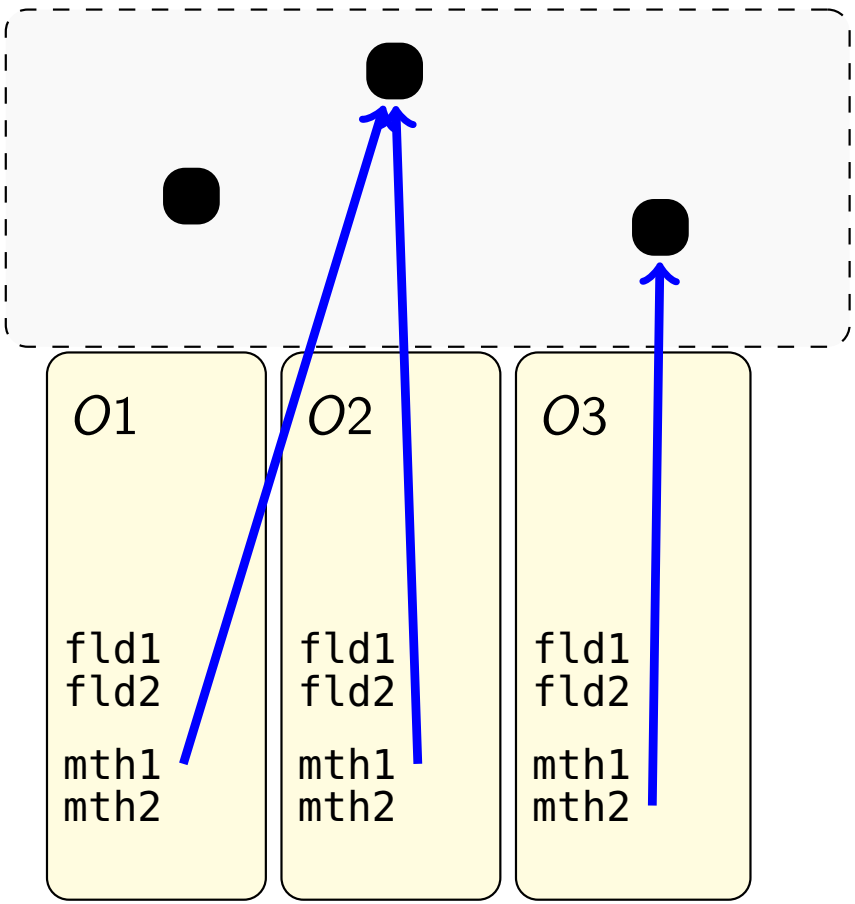
- global heap
- active objects
- shared resources
- concurrent access

Overview



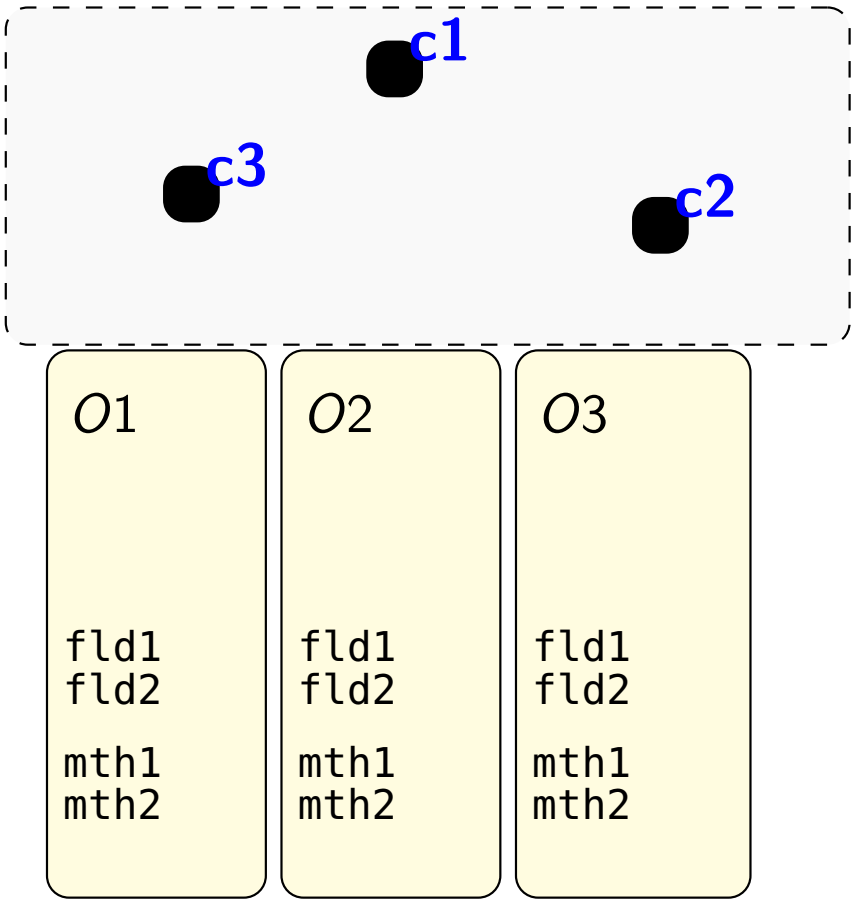
- global heap
- active objects
- shared resources
- concurrent access

Overview



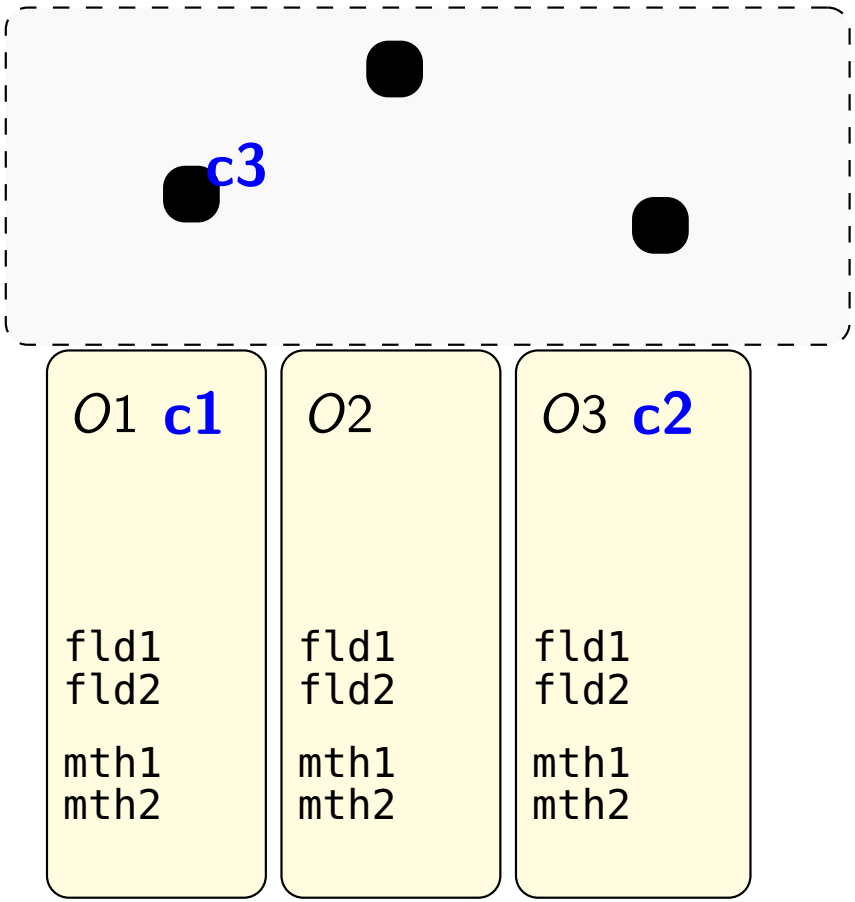
- global heap
- active objects
- shared resources
- concurrent access
- race conditions

Overview



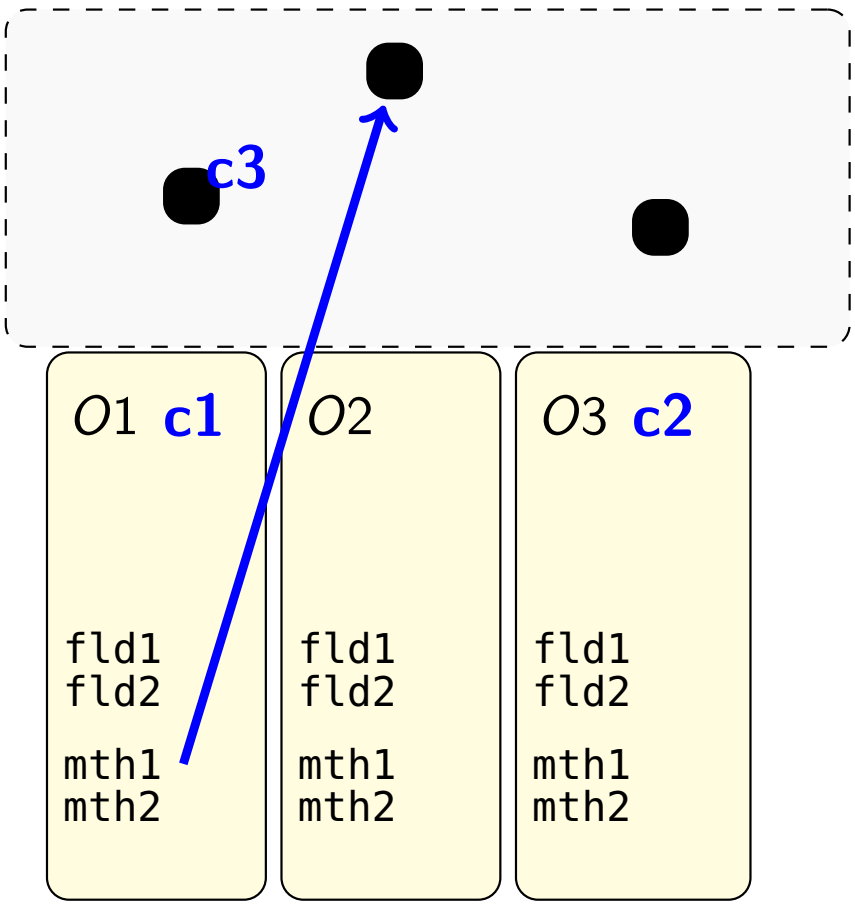
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access

Overview



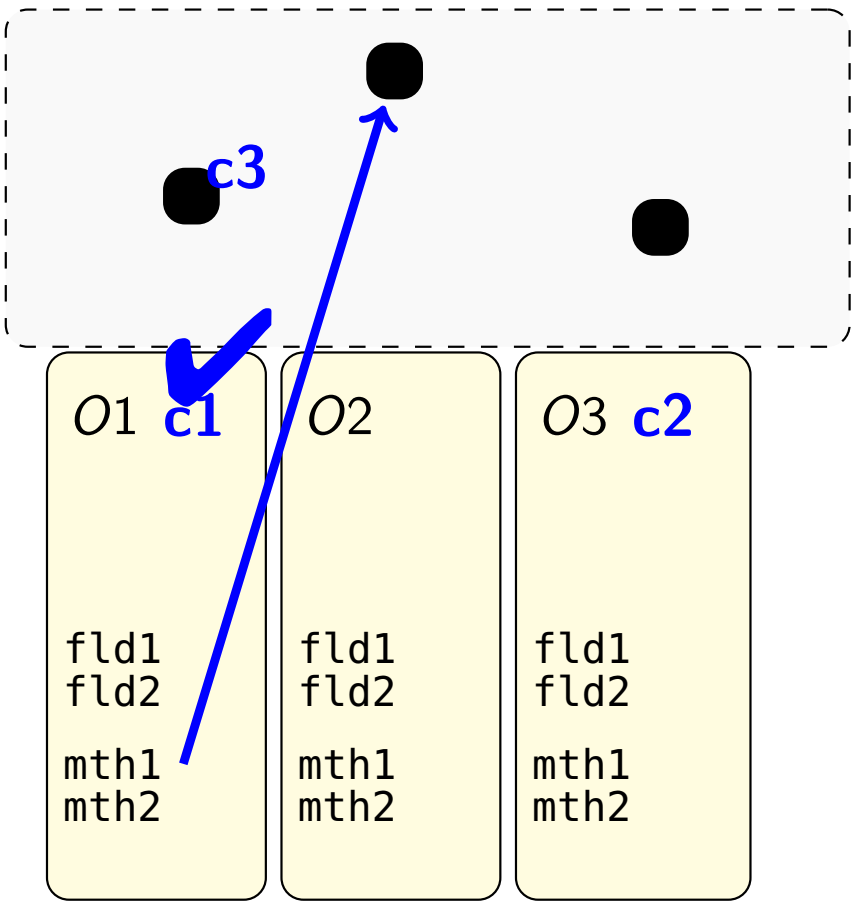
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities

Overview



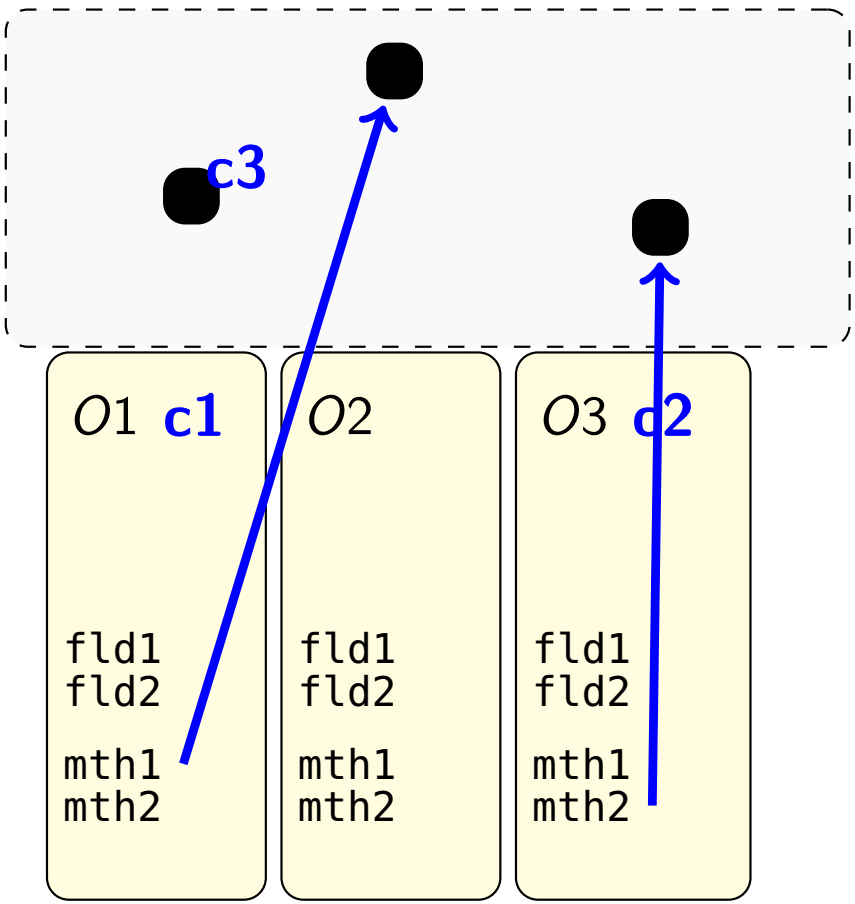
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability

Overview



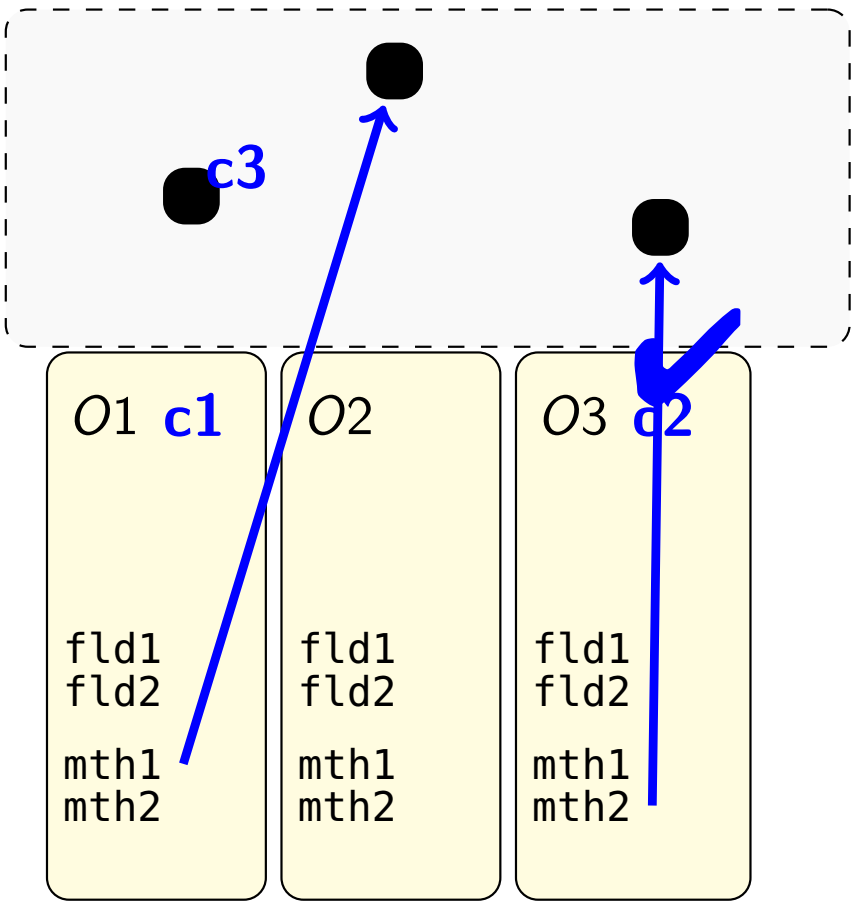
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability

Overview



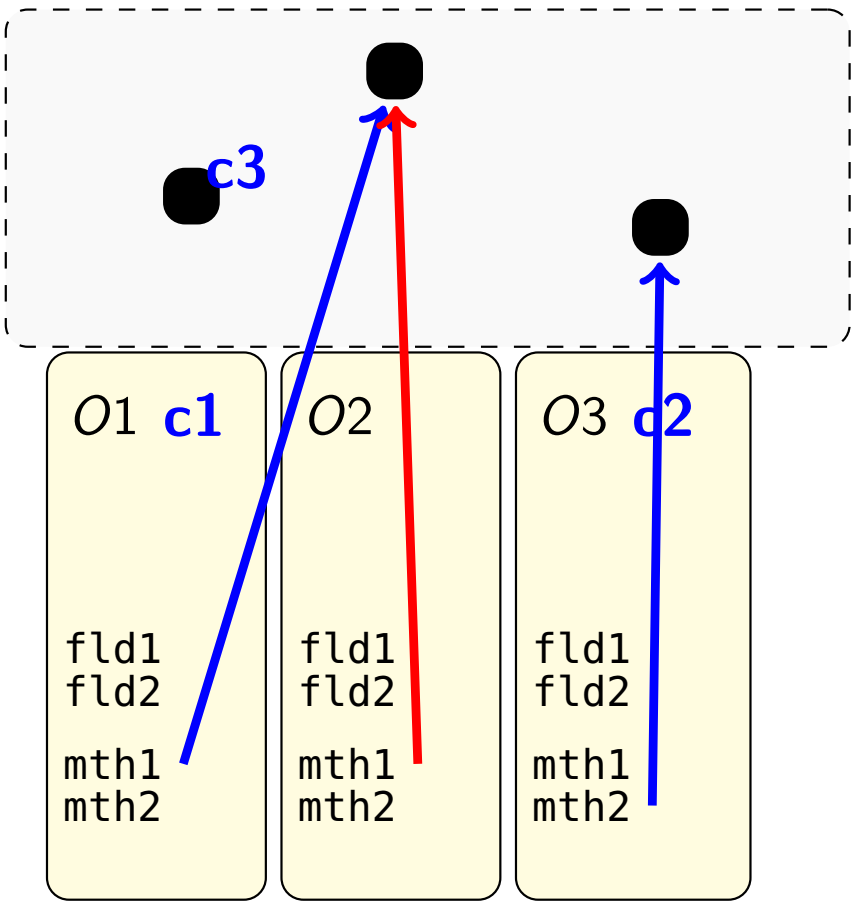
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability

Overview



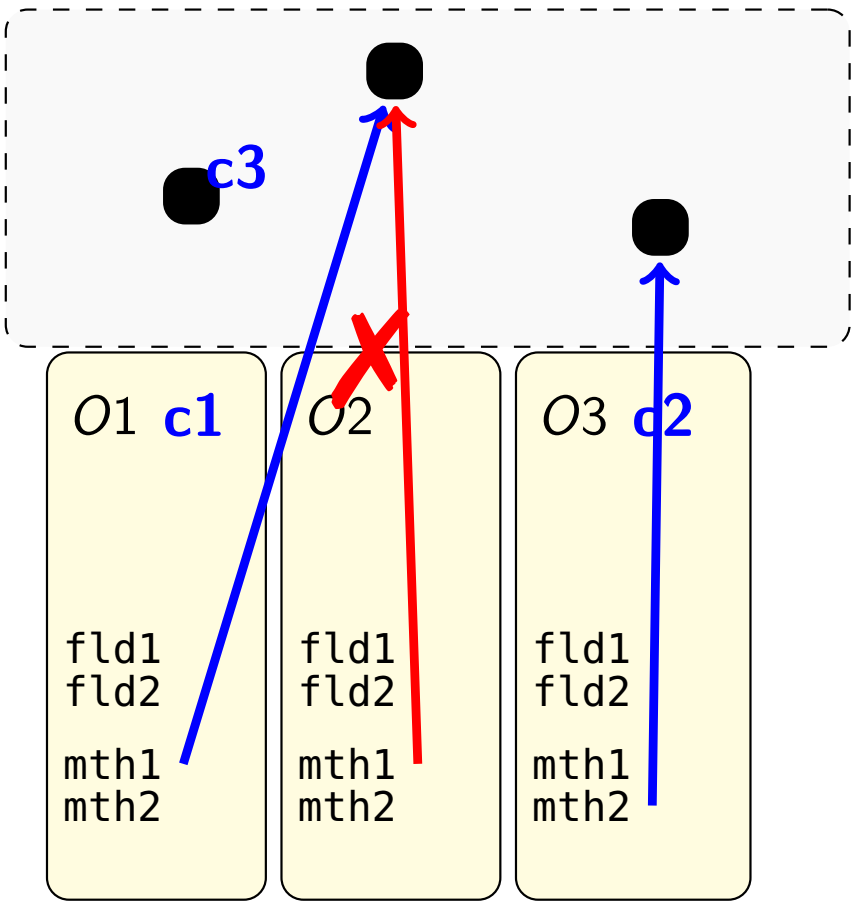
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability

Overview



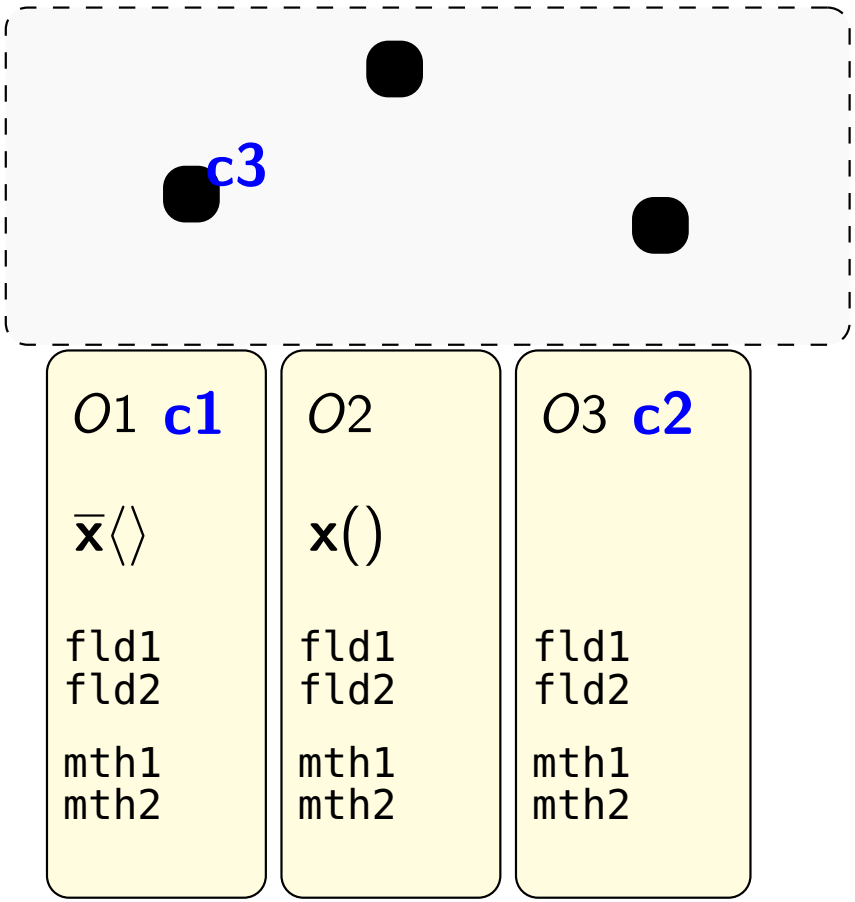
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability

Overview



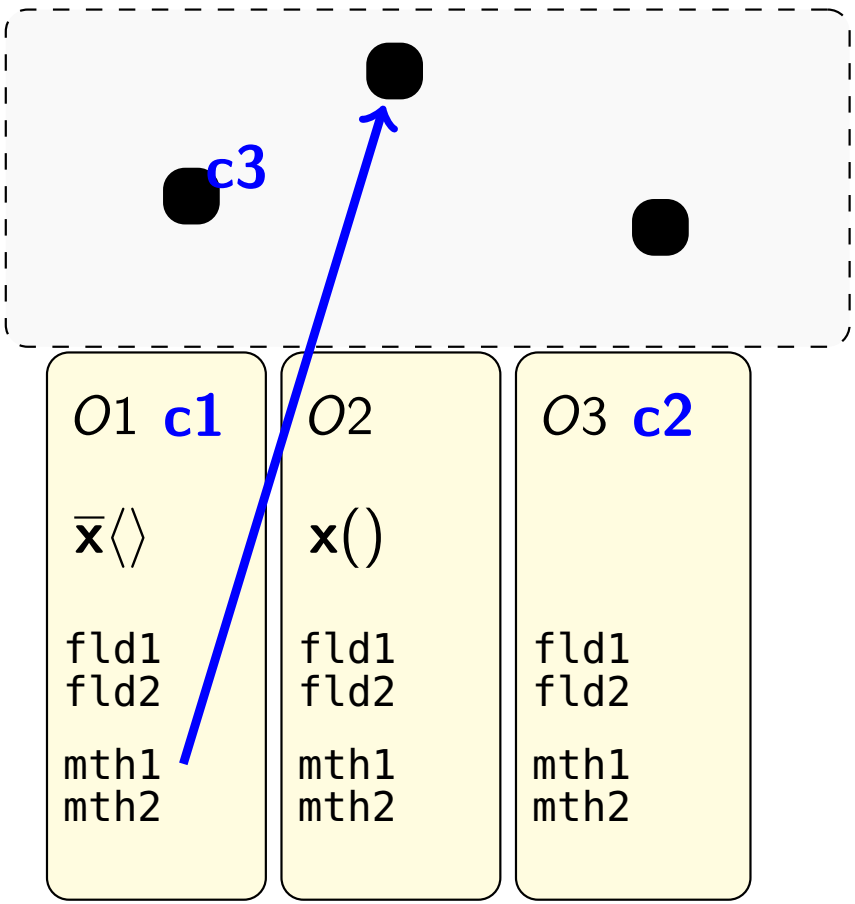
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability

Overview



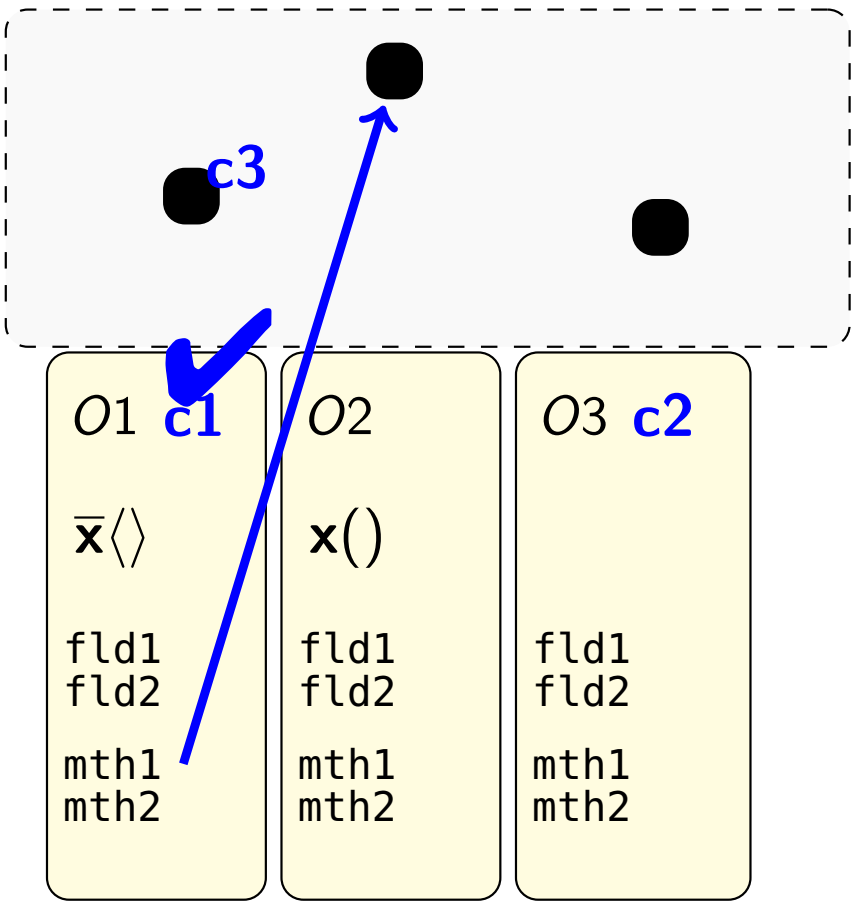
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels

Overview



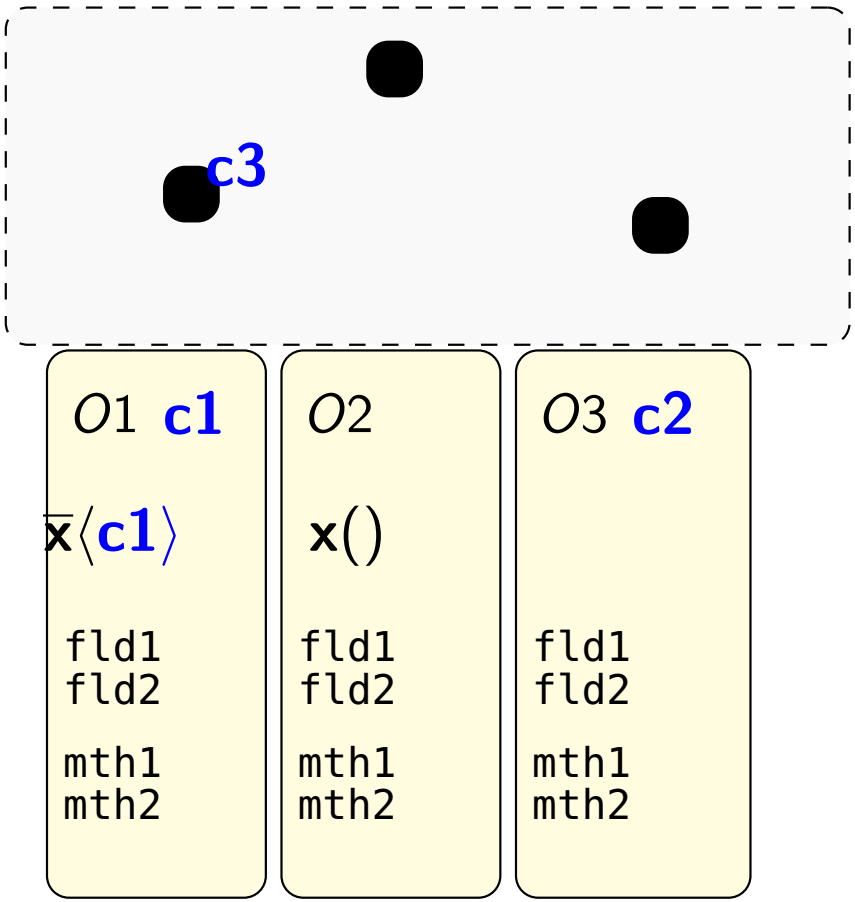
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Overview



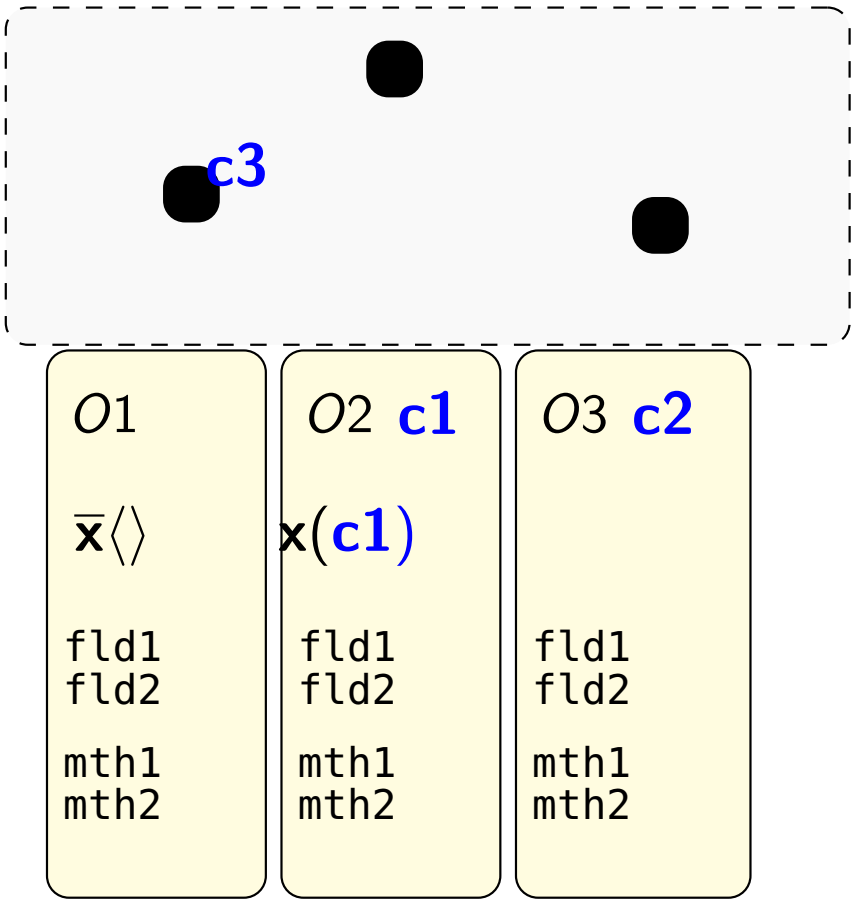
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Overview



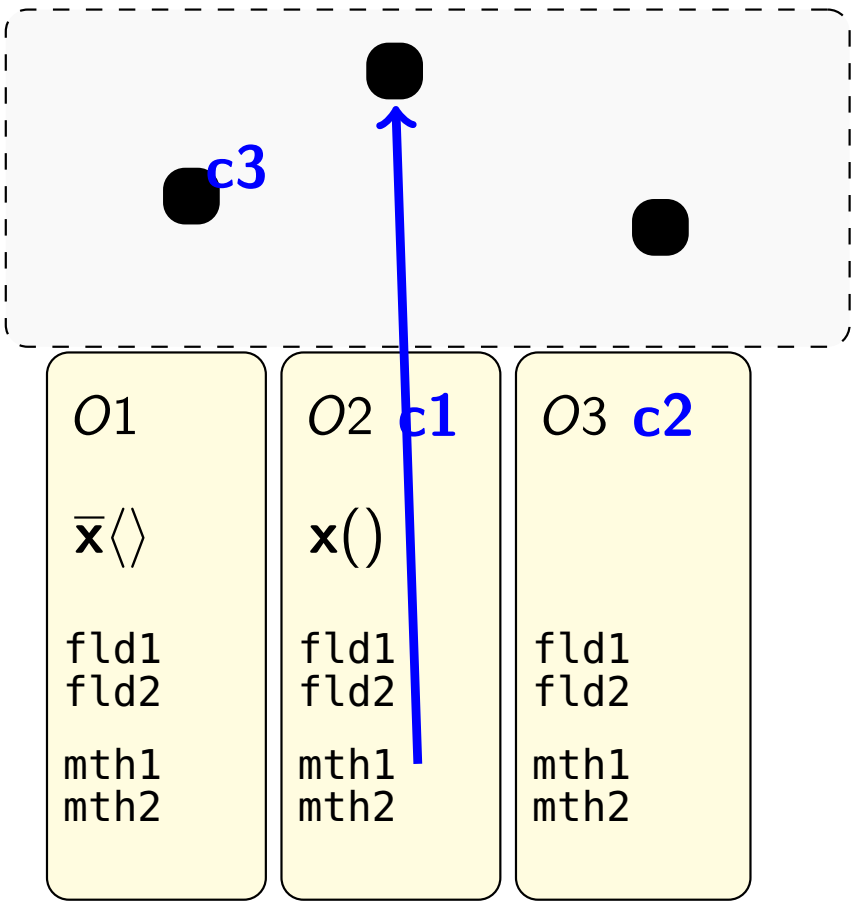
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Overview



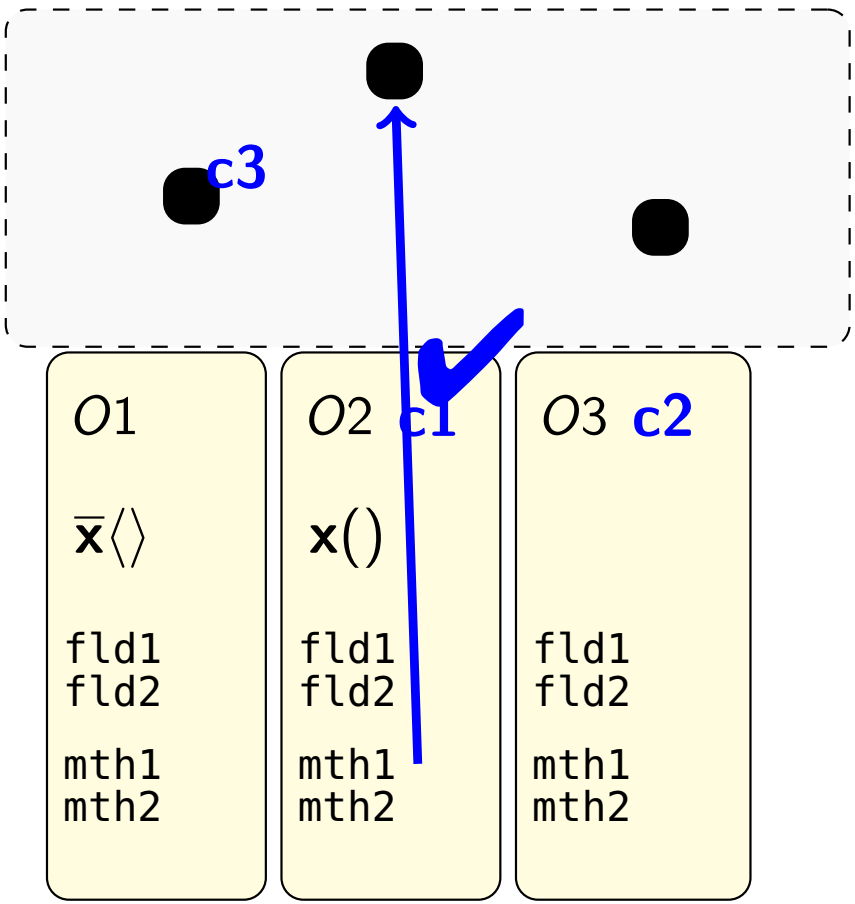
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Overview



- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Overview



- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Example (bad)

Two processes P and Q write (W) to a shared resource (*file*).

Example (bad)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file : R)(P \mid Q)$$

$$P = W(file).\emptyset \qquad Q = W(file).\emptyset$$

Example (bad)

Two processes P and Q write (W) to a shared resource ($file$).

$$SYS = (\nu file : R)(P \mid Q)$$

$$P = W(file).\emptyset \qquad Q = W(file).\emptyset$$

- race condition!

Example (bad)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file : R)(P \mid Q)$$

$$P = W(file).\emptyset \qquad Q = W(file).\emptyset$$

- race condition!
  1. we do not want any of those

Example (bad)

Two processes P and Q write (W) to a shared resource ($file$).

$$SYS = (\nu file : R)(P \mid Q)$$

$$P = W(file).\emptyset \qquad Q = W(file).\emptyset$$

- race condition!
  1. we do not want any of those
  2. we want to reason about concurrent operations on resources

Example (good)

Two processes P and Q write (W) to a shared resource (*file*).

Example (good)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \qquad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

Example (good)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \qquad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists\alpha R.!W(\alpha).\mathsf{end}$$

Example (good)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \qquad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists\alpha R.!W(\alpha).\mathsf{end}$$

- well-typed

Example (good)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (\ W(file)\ .x\langle file\rangle.x\langle wr\rangle.\emptyset) \qquad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists\alpha R.!W(\alpha).\mathsf{end}$$

- well-typed
- $W(file)$ is ok, P is typed against wr, Q is not

Example (good)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\ \emptyset\ ) \qquad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists\alpha R.!W(\alpha).\mathsf{end}$$

- well-typed
- $W(file)$ is ok, P is typed against wr, Q is not
- $\emptyset$ in P is typed without wr (was sent in $x\langle wr\rangle$)

Example (good)

Two processes P and Q write (W) to a shared resource $(file)$.

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \qquad Q = (\ y(fvar)\ .y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists\ \alpha\ R.!W(\ \alpha\ ).\text{end}$$

- well-typed
- $W(file)$ is ok, P is typed against wr, Q is not
- $\emptyset$ in P is typed without wr (was sent in $x\langle wr\rangle$)
- $y(fvar)$ binds α to $fvar$ in the capability received later in S

Example (good)

Two processes P and Q write (W) to a shared resource ($file$).

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \qquad Q = (y(fvar).y(wrvar).\ W(fvar)\ .\emptyset)$$

$$S = !\exists\alpha R.!W(\alpha).\text{end}$$

- well-typed
- $W(file)$ is ok, P is typed against wr, Q is not
- $\emptyset$ in P is typed without wr (was sent in $x\langle wr\rangle$)
- $y(fvar)$ binds α to $fvar$ in the capability received later in S
- so $W(fvar)$ in Q succeeds

Reduction

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$
$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset)$$
$$Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q[file/fvar] \xrightarrow{tau} P \mid Q[wr/wrvar] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset$$

Reduction

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (\ W(file)\ .x\langle file\rangle.x\langle wr\rangle.\emptyset)$$

$$Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q[file/fvar] \xrightarrow{tau} P \mid Q[wr/wrvar] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset$$

- $W(file)$ happens in P

Reduction

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).\ x\langle file\rangle.x\langle wr\rangle.\emptyset)$$

$$Q = (\ y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q\ [file/fvar] \xrightarrow{tau} P \mid Q[wr/wrvar] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset$$

- $W(file)$ happens in P
- exchange of the resource

Reduction

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.\ x\langle wr\rangle\ .\emptyset)$$

$$Q = (y(fvar).\ y(wrvar)\ .W(fvar).\emptyset)$$

$$SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q[file/fvar] \xrightarrow{tau} P \mid Q\ [wr/wrvar] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset$$

- $W(file)$ happens in P
- exchange of the resource and of the wr capability

Reduction

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$
$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset)$$
$$Q = (y(fvar).y(wrvar).\ W(fvar)\ .\emptyset)$$

$$SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q[file/fvar] \xrightarrow{tau} P \mid Q[wr/wrvar] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset$$

- $W(file)$ happens in P
- exchange of the resource and of the wr capability
- $W(file)$ happens in Q

Idea

- core calculus (π-calculus inspired)
- session types regulate sharing of capabilities
- capabilities are abstract and linear
- well-typed processes have no race conditions (for some def. of race condition)

Syntax of processes

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y \rangle.P \\
\quad \mid x(y).P & \mid x \lhd a.P & \mid x.\{a_i.P_i\}_{i \in I} \\
\quad \mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : R)P & \mid \varphi(r).P
\end{array}
$$

Syntax of processes

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y \rangle.P \\
\quad \mid x(y).P & \mid x \lhd a.P & \mid x.\{a_i.P_i\}_{i \in I} \\
\quad \mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : R)P & \mid \varphi(r).P
\end{array}
$$

- standard π-calculus + sessions (Vasconcelos)

Syntax of processes

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y \rangle.P \\
\quad \mid x(y).P & \mid x \triangleleft a.P & \mid x.\{a_i.P_i\}_{i \in I} \\
\quad \mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : \mathrm{R})\, P & \mid \varphi(r).P
\end{array}
$$

- standard π-calculus + sessions (Vasconcelos)
- co-creation of resource r and capabilities $\overline{c}$
- capabilities have type σ

Syntax of processes

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y \rangle.P \\
\quad \mid x(y).P & \mid x \lhd a.P & \mid x.\{a_i.P_i\}_{i \in I} \\
\quad \mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : R)P & \mid \varphi(r).P
\end{array}
$$

- standard π-calculus + sessions (Vasconcelos)
- co-creation of resource r and capabilities $\overline{c}$
- capabilities have type σ
- (abstract) operation on resource r

Syntax of processes and types

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y \rangle.P \\
\mid x(y).P & \mid x \triangleleft a.P & \mid x.\{a_i.P_i\}_{i \in I} \\
\mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : R)P & \mid \varphi(r).P
\end{array}
$$

$$
\begin{array}{llll}
S ::= !t.S & \mid \&\{a_i : S_i\}_{i \in I} & t ::= S & \sigma ::= \varphi(r) \\
\mid ?t.S & \mid \oplus\{a_i : S_i\}_{i \in I} & \mid \sigma & \mid \varphi(\alpha) \\
\mid \mathsf{end} & & \mid R & \\
 & & \mid \exists \alpha R &
\end{array}
$$

Syntax of processes and types

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y\rangle.P \\
\mid x(y).P & \mid x \triangleleft a.P & \mid x.\{a_i.P_i\}_{i\in I} \\
\mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : R)P & \mid \varphi(r).P
\end{array}
$$

$$
\begin{array}{llll}
S ::= !t.S & \mid \&\{a_i : S_i\}_{i\in I} & t ::= S & \sigma ::= \boxed{\varphi(r)} \\
\mid ?t.S & \mid \oplus\{a_i : S_i\}_{i\in I} & \mid \sigma & \mid \boxed{\varphi(\alpha)} \\
\mid \text{end} & & \mid R & \\
 & & \mid \exists\alpha R &
\end{array}
$$

- capability types specify what (abstract) operation can be executed on a resource

Syntax of processes and types

$$
\begin{array}{lll}
P ::= \emptyset & \mid P \mid P & \mid x\langle y\rangle.P \\
\mid x(y).P & \mid x \triangleleft a.P & \mid x.\{a_i.P_i\}_{i\in I} \\
\mid (\nu xy : S)P & \mid (\nu r^{\overline{c}:\overline{\sigma}} : R)P & \mid \varphi(r).P
\end{array}
$$

$$
\begin{array}{llll}
S ::= !t.\ S & \mid \&\{a_i : S_i\}_{i\in I} & t ::= S & \sigma ::= \varphi(r) \\
\mid ?t.\ S & \mid \oplus\{a_i : S_i\}_{i\in I} & \mid \sigma & \mid \varphi(\alpha) \\
\mid \text{end} & & \mid R & \\
 & & \mid \exists\ \alpha\ R &
\end{array}
$$

- capability types specify what (abstract) operation can be executed on a resource
- existential quantifiers bind resource variables α in the continuation of the session

Properties

- progress: $\vdash P$ and $P \rightarrow P'$ implies $\vdash P'$

Properties

- progress: $\vdash P$ and $P \rightarrow P'$ implies $\vdash P'$
- no errors:

Properties

- progress: $\vdash P$ and $P \rightarrow P'$ implies $\vdash P'$
- no errors:
  - no race conditions: $\vdash P$ implies $P \not\rightarrow P'$ and $P' \equiv \varphi(r).Q | \varphi'(r').Q'$ where $\varphi(r) \not\simeq \varphi'(r')$

    Where $\varphi(r) \simeq \varphi'(r')$
    if $\varphi(r).\varphi'(r') = \varphi'(r')\varphi(r)$
    (i.e. the order of execution of $\varphi(r)$ and $\varphi'(r')$ does not matter)

So far, this is very much WIP

- borrowing capabilities (auto get back after some time)

So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system

So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
- add locations (model group of resources as ownership)

So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
- add locations (model group of resources as ownership) and capabilities on locations

So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
- add locations (model group of resources as ownership) and capabilities on locations
- more?