An Asynchronous Soundness Theorem for Concurrent Separation Logic

Paul-André Melliès

Léo Stefanesco

IRIF, CNRS & Université Paris Diderot

Séminaire Gallium

4 juin 2018

Summary

- **The imperative concurrent language** and its semantics

$$[\![C]\!]_S \qquad\qquad [\![C]\!]_L$$

- **Concurrent separation Logic** and its semantics

$$\left[\!\!\left[\frac{\vdots\, \pi}{\Gamma \vdash \{P\} C \{Q\}} \right]\!\!\right]_{Sep}$$

Summary

- **The imperative concurrent language** and its semantics

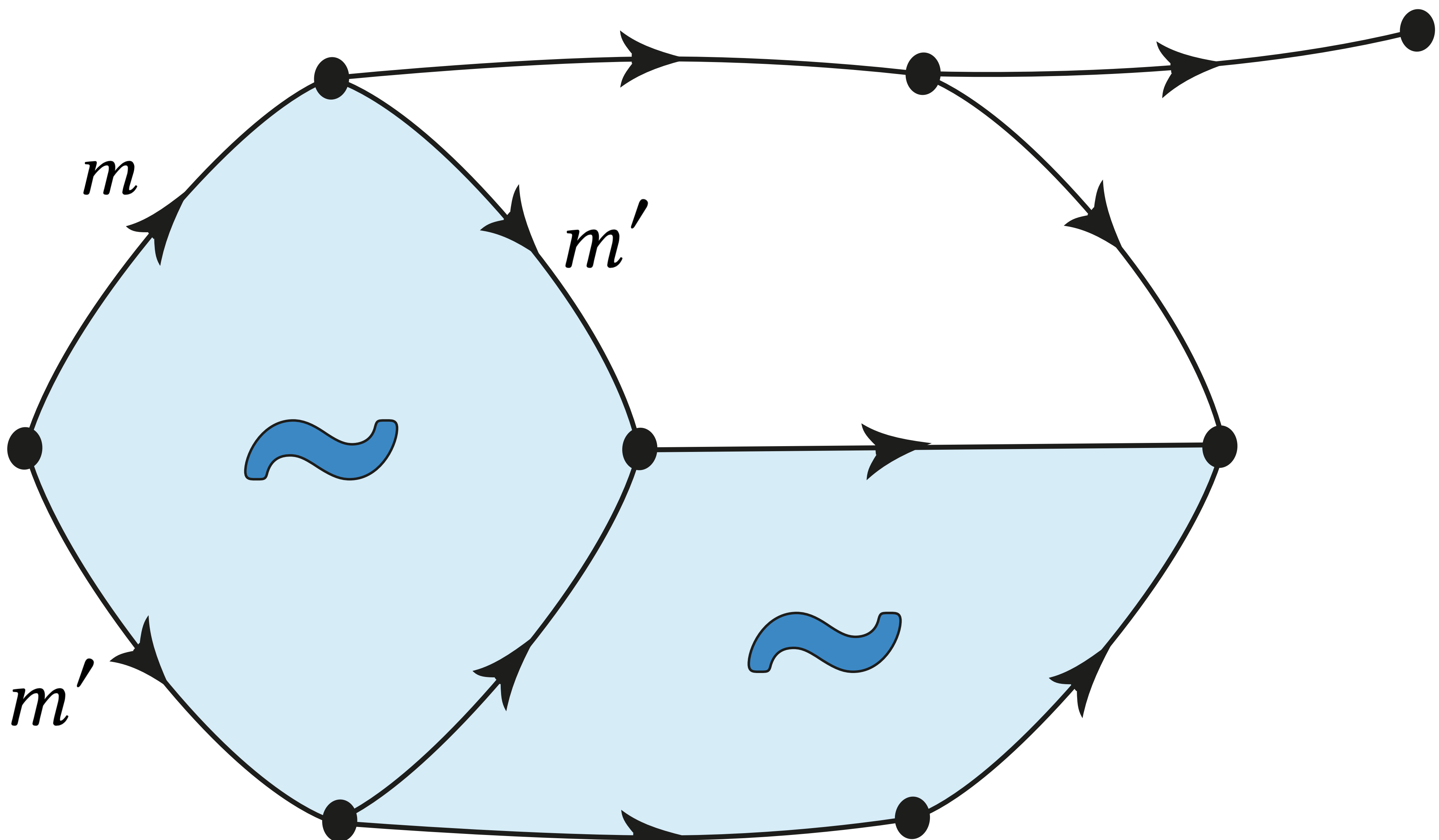
$$[\![C]\!]_S \xrightarrow{\quad\mathscr{L}\quad} [\![C]\!]_L$$

- **Concurrent separation Logic** and its semantics

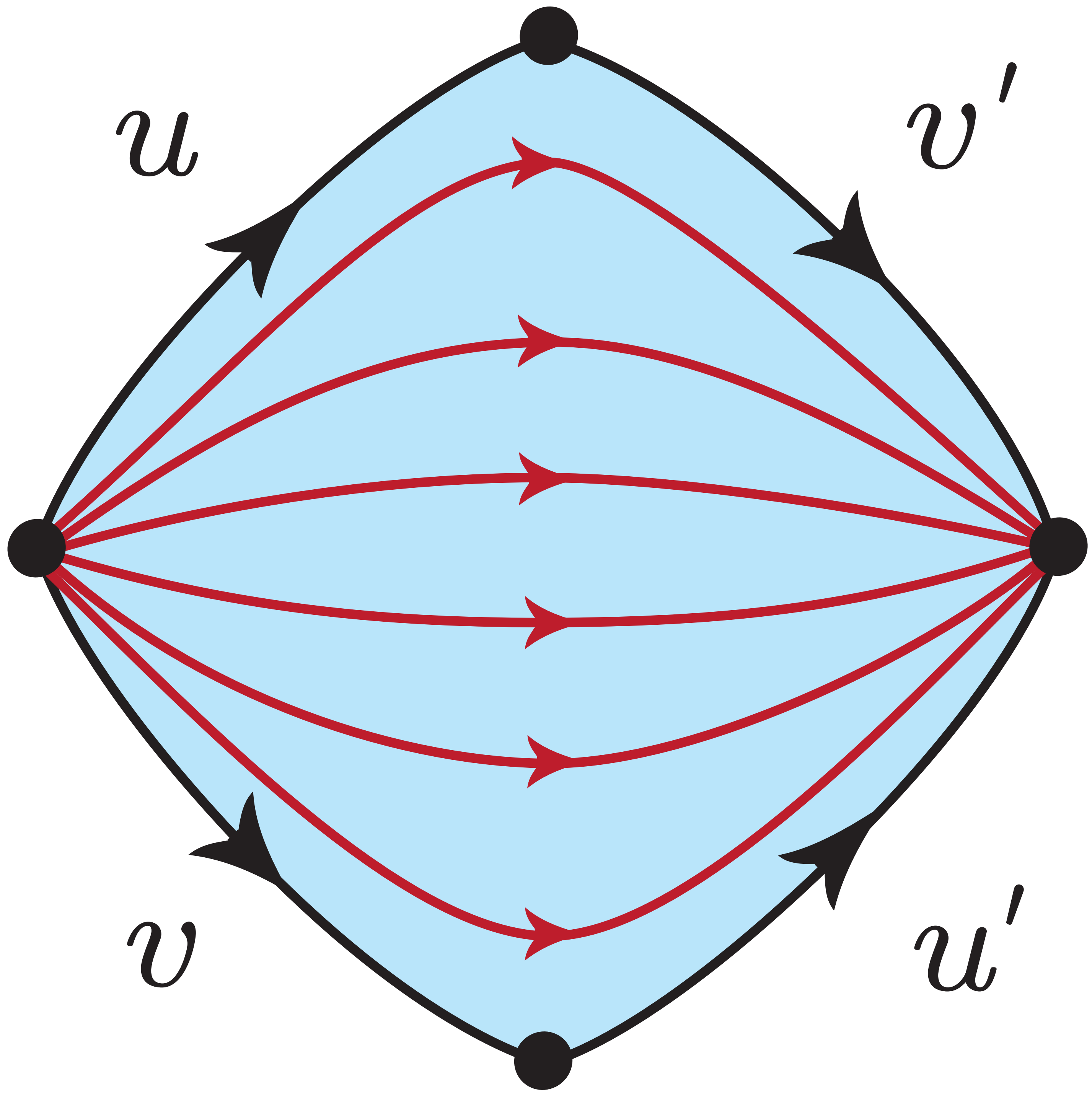
$$\left[\!\!\left[\begin{array}{c}\vdots\,\pi\\ \hline \Gamma \vdash \{P\}C\{Q\}\end{array}\right]\!\!\right]_{Sep} \xrightarrow{\quad\mathscr{S}\quad} [\![C]\!]_S$$

- **Soundness theorems**: relating those semantics

Asynchronous Graphs

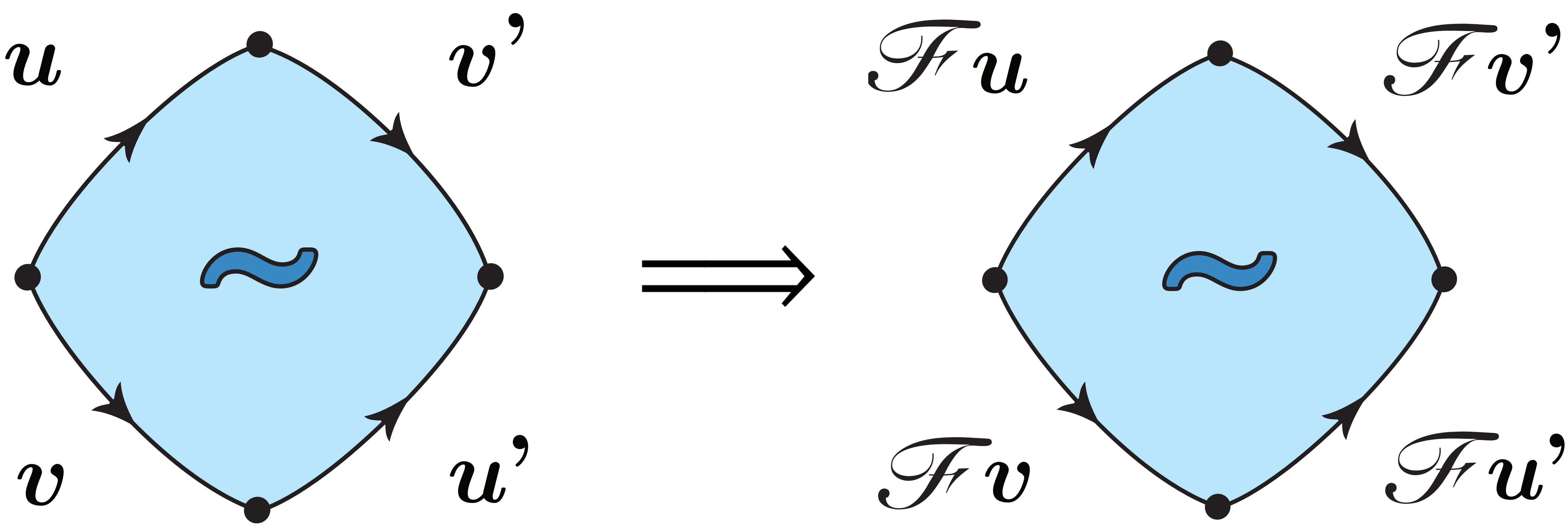


Topological Intuition

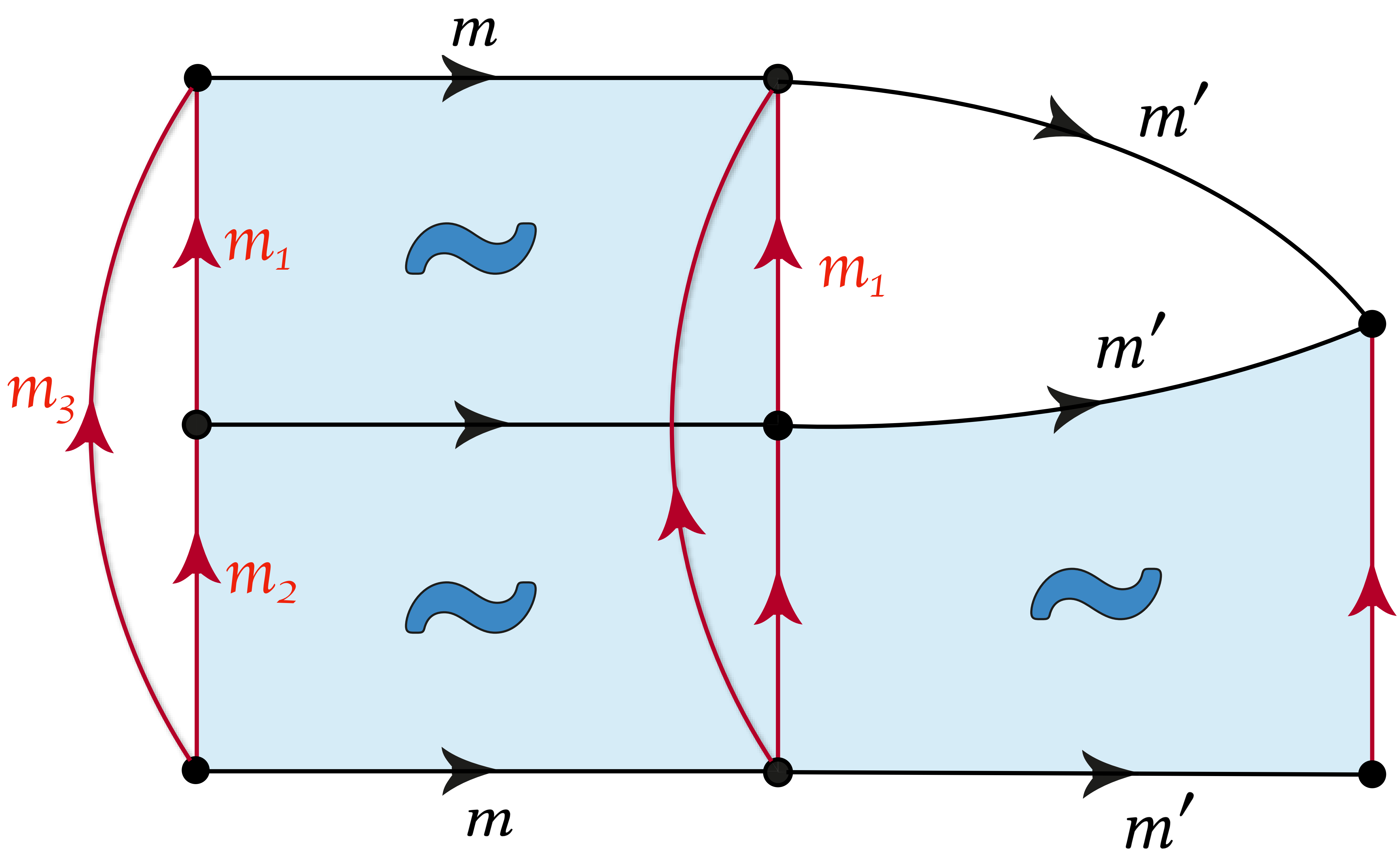


Asynchronous Morphisms

It’s a graph homomorphism, such that:



Asynchronous Graphs with Environment



A Simple Concurrent Language

$$
\begin{array}{l}
\text{resource } r \text{ do} \\
\quad \text{while } x > 0 \text{ do} \\
\quad \left\{ \begin{array}{c}
x := x - 1; \\
\begin{array}{l} \text{with } r \text{ do} \\ \quad y := y + 1 \end{array} \;\Big\|\; \begin{array}{l} \text{with } r \text{ do} \\ \quad y := y + 1 \end{array}
\end{array} \right\}
\end{array}
$$

A Simple Concurrent Language

Source language

$$
\begin{aligned}
B &::= \mathbf{true} \mid \mathbf{false} \mid B \wedge B' \mid B \vee B' \mid E = E' \\
E &::= 0 \mid 1 \mid \ldots \mid x \mid E + E' \mid E * E' \\
C &::= x := E \mid x := [E] \mid [E] := E' \mid C; C' \mid C_1 \parallel C_2 \mid \texttt{skip} \\
&\quad \mid \texttt{while}\ B\ \texttt{do}\ C \mid \texttt{resource}\ r\ \texttt{do}\ C \mid \texttt{with}\ r\ \texttt{when}\ B\ \texttt{do}\ C \\
&\quad \mid \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \mid x := \texttt{alloc}(E) \mid \texttt{dispose}(E)
\end{aligned}
$$

“Assembly language”

$$
\begin{aligned}
m ::= {} & x := E \mid x := [E] \mid [E] := E' \mid \texttt{nop} \\
& \mid x := \texttt{alloc}(E) \mid \texttt{dispose}(E) \mid P(r) \mid V(r)
\end{aligned}
$$

State transitions

$$(\mu, L) \overset{m}{\rightsquigarrow} (\mu', L') \qquad\qquad (\mu, L) \overset{m}{\rightsquigarrow} \lightning$$

where μ is the memory and L is the set of held locks

Asynchronous Transition System

$$\lambda \quad : \quad G \longrightarrow \mathcal{T}$$

Code-acyclic asynchronous graph

Machine model

$$\lambda_{[\![C]\!]_S} : [\![C]\!]_S \longrightarrow \mathcal{T}_S$$

$$\lambda_{[\![C]\!]_L} : [\![C]\!]_L \longrightarrow \mathcal{T}_L$$

Asynchronous Transition System

$$\lambda \quad : \quad G \longrightarrow \text{ナ}$$

Code-acyclic asynchronous graph

Machine model

Two semantics:

$$\lambda_{[\![C]\!]_S} : [\![C]\!]_S \longrightarrow \text{ナ}_S$$

$$\lambda_{[\![C]\!]_L} : [\![C]\!]_L \longrightarrow \text{ナ}_L$$

Asynchronous Transition System

$$\lambda \quad : \quad G \longrightarrow \text{⚓}$$

Code-acyclic asynchronous graph

Machine model

Two semantics:

$$\lambda_{[\![C]\!]_S} : [\![C]\!]_S \longrightarrow \text{⚓}_S$$

$$\lambda_{[\![C]\!]_L} : [\![C]\!]_L \longrightarrow \text{⚓}_L$$

λ is an Environment 1-fibration = “the Environment can always execute every instructions”

Machine Models for the Code

$ㄊ_S$

Asynchronous graph of machine states

Nodes: $\mathfrak{s} = (\mu, L)$

Edges: $\mathfrak{s} \overset{m}{\rightsquigarrow} \mathfrak{s}'$

$$\begin{tikzcd} & \mathfrak{s}_2' \arrow[dr, "m'"] & \\ \mathfrak{s}_1 \arrow[ur, "m"] \arrow[dr, "m'"'] & & \mathfrak{s}_3 \\ & \mathfrak{s}_2 \arrow[ur, "m"'] & \end{tikzcd}$$

is a tile when:

$$(\mathrm{rd}(m) \cup \mathrm{wr}(m)) \cap \mathrm{wr}(m') = \emptyset$$
$$(\mathrm{rd}(m') \cup \mathrm{wr}(m')) \cap \mathrm{wr}(m) = \emptyset$$
$$\mathrm{lock}(m) \cap \mathrm{lock}(m') = \emptyset$$

$ㄊ_L$

Asynchronous graph of locks

Nodes: L

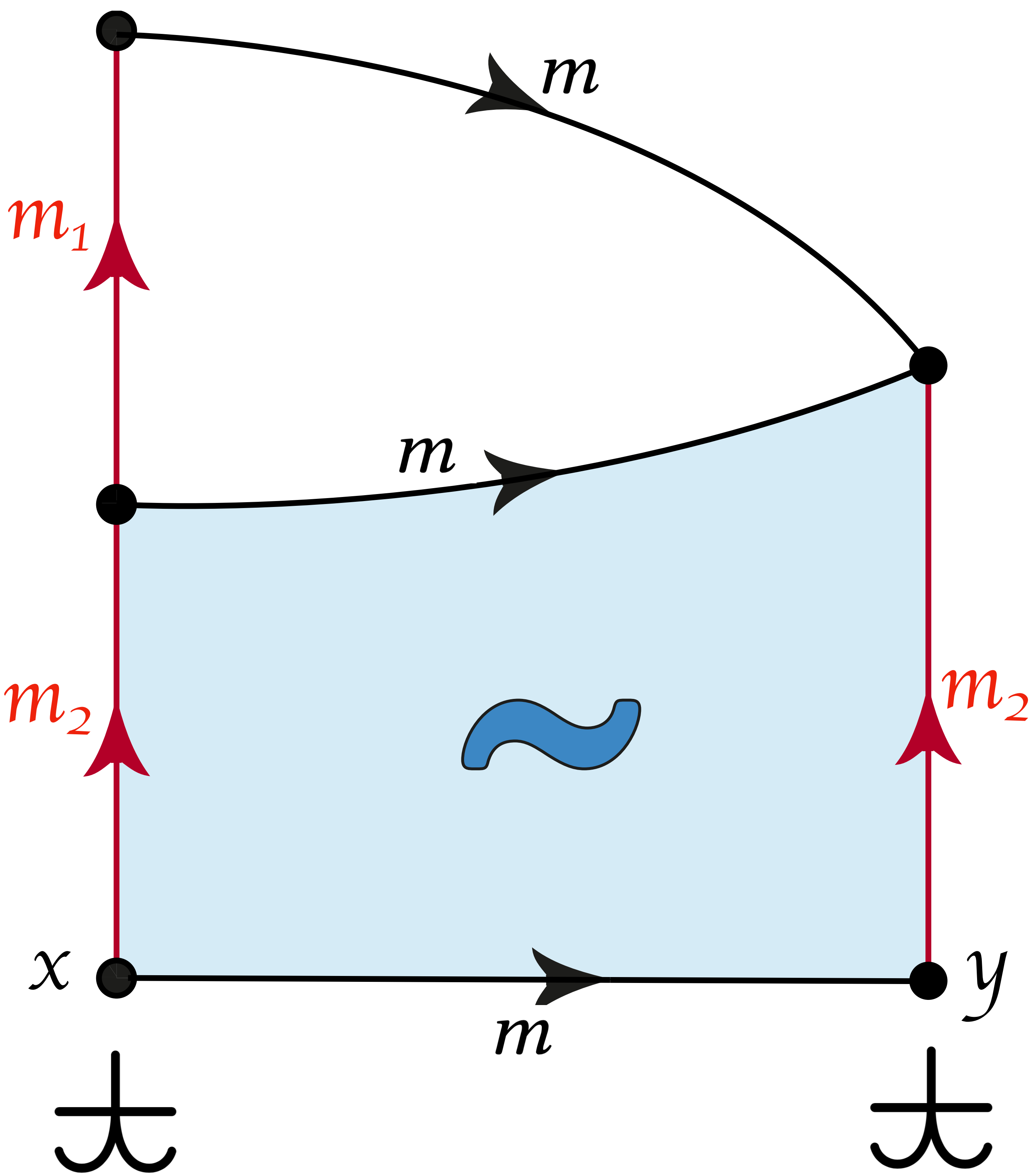
Edges: $L \overset{m}{\rightsquigarrow} L'$

$$\begin{tikzcd} & L_2' \arrow[dr, "m'"] & \\ L_1 \arrow[ur, "m"] \arrow[dr, "m'"'] & & L_3 \\ & L_2 \arrow[ur, "m"'] & \end{tikzcd}$$

is a tile when:

$$\mathrm{lock}(m) \cap \mathrm{lock}(m') = \emptyset$$

Semantics of leaves



such that:

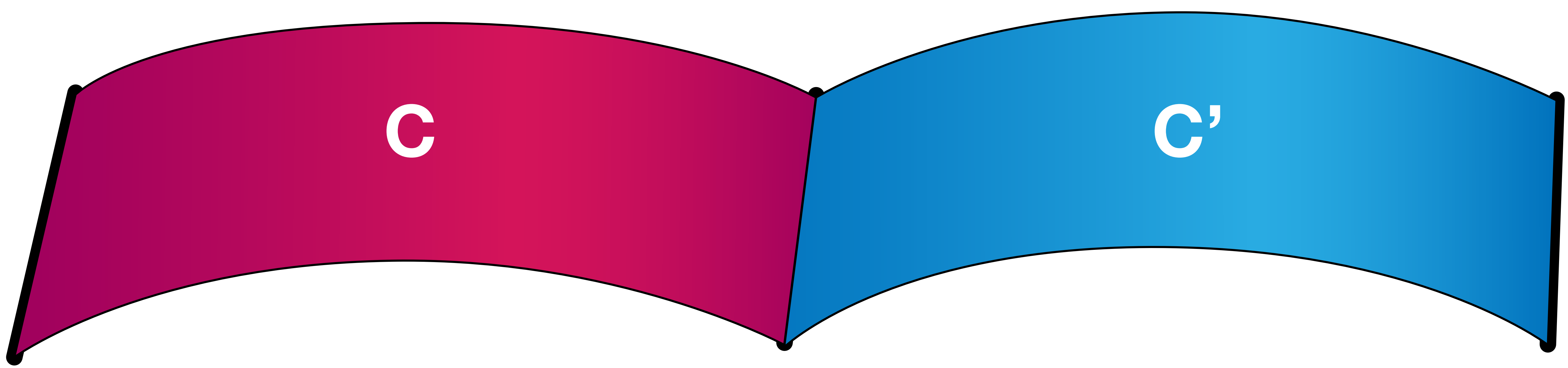
$$\lambda(x) \overset{m}{\rightsquigarrow} \lambda(y)$$

There is a tile whenever
the footprints are independent

Graphical representation

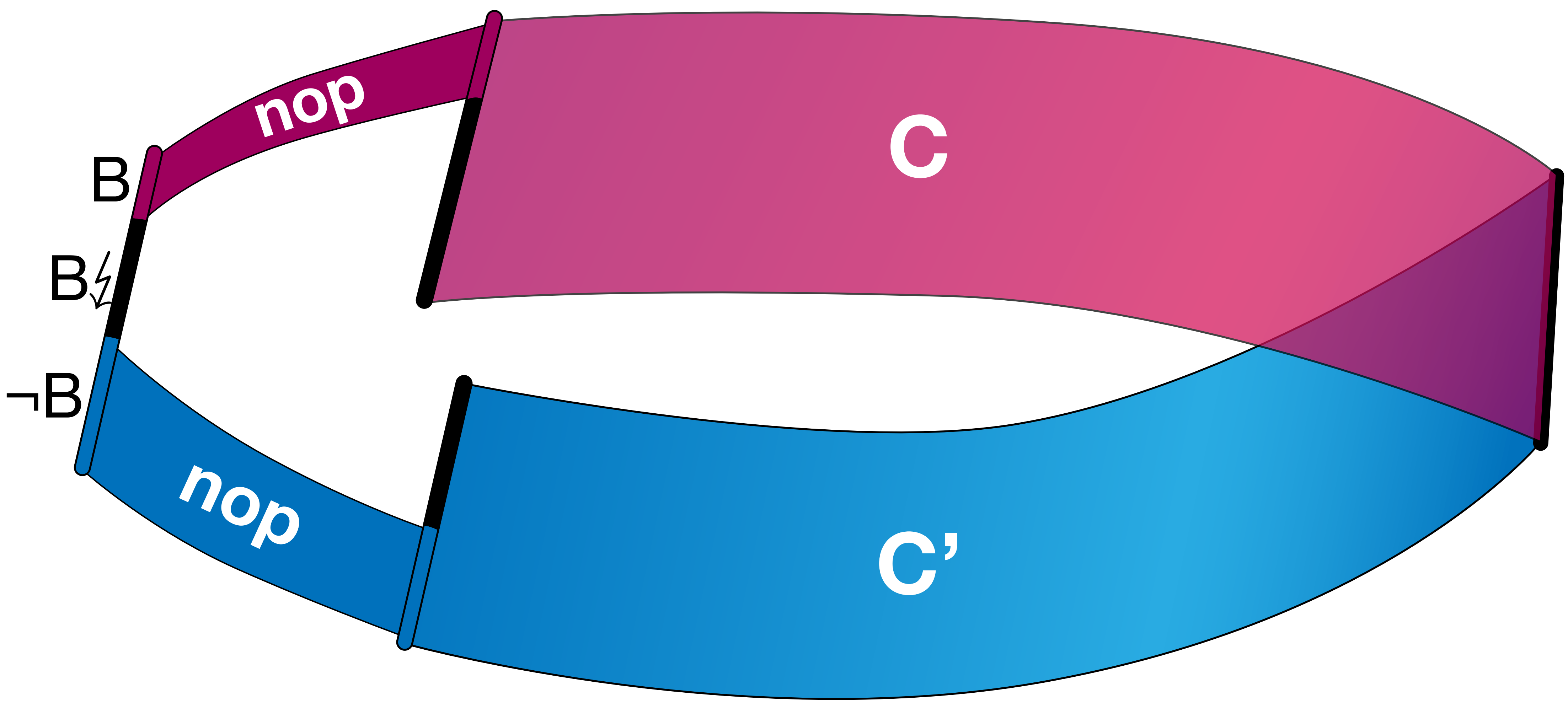
Sequential Composition

$$[\![C ; C']\!] \ = \ [\![C]\!] ; [\![C']\!]$$



Conditionals

$$
\begin{aligned}
[\![\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C']\!] = {} & \mathsf{whentrue}[B]([\![\mathsf{nop}]\!]); [\![C]\!] \\
& \oplus \mathsf{whenfalse}[B]([\![\mathsf{nop}]\!]); [\![C']\!] \\
& \oplus \mathsf{whenabort}[B]
\end{aligned}
$$



Parallel Product: $G_1 \parallel G_2$

Nodes: $x_1|x_2 \in G_1{\times}G_2$ such that $\lambda_{G_1}(x_1) = \lambda_{G_2}(x_2)$

Parallel Product: $G_1 \parallel G_2$

Nodes: $x_1|x_2 \in G_1 \times G_2$ such that $\lambda_{G_1}(x_1) = \lambda_{G_2}(x_2)$

Edges: an edge $x_1|x_2 \xrightarrow{\quad m \,|\, m \quad} x_1'|x_2'$ is a pair of edges

$x_1 \xrightarrow{\quad m \quad} x_1'$ in G_1 and $x_2 \xrightarrow{\quad m \quad} x_2'$ in G_2

Parallel Product: $G_1 \parallel G_2$

Nodes: $x_1|x_2 \in G_1 \times G_2$ such that $\lambda_{G_1}(x_1) = \lambda_{G_2}(x_2)$

Edges: an edge $x_1|x_2 \xrightarrow{m \,|\, m} x_1'|x_2'$ is a pair of edges

$x_1 \xrightarrow{m} x_1'$ in G_1 and $x_2 \xrightarrow{m} x_2'$ in G_2

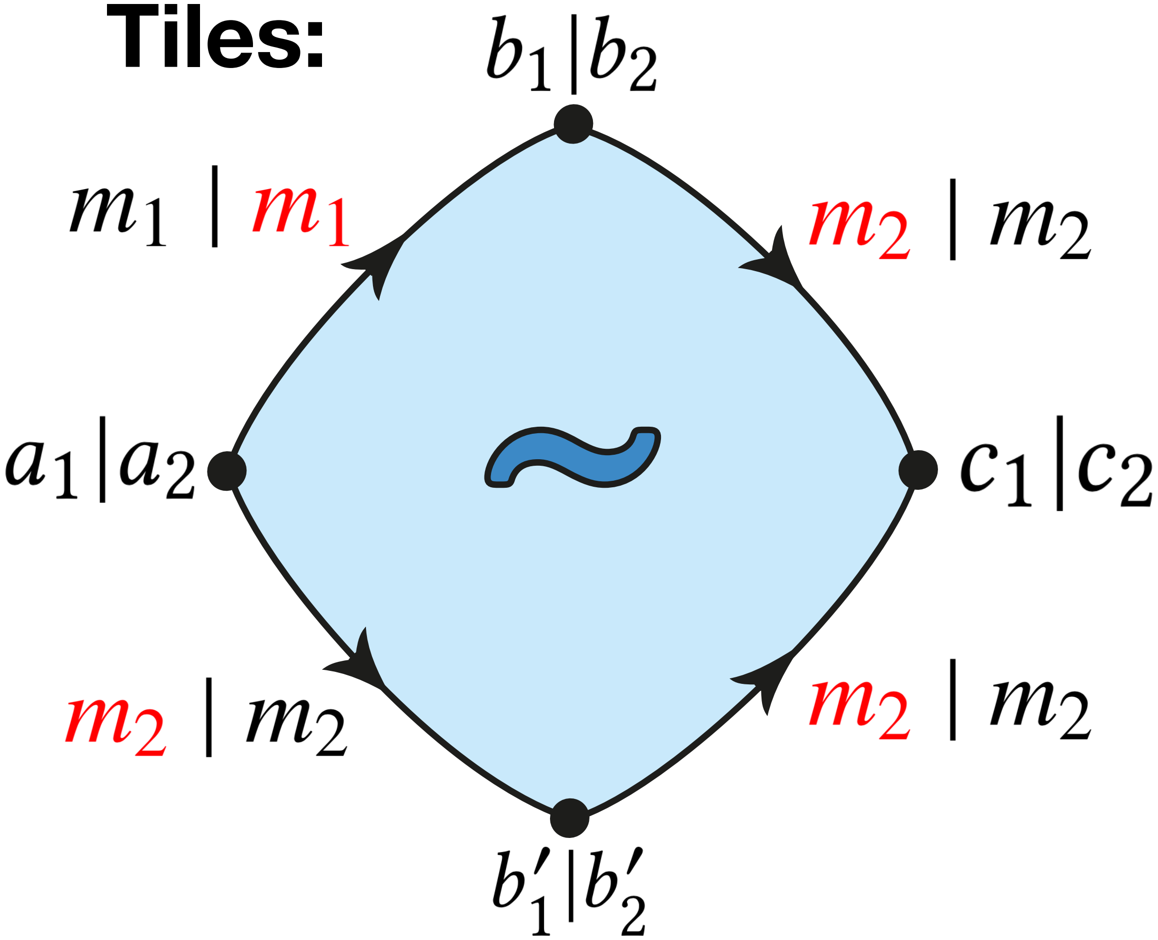
Parallel Product: $G_1 \parallel G_2$

Nodes: $x_1 | x_2 \in G_1 \times G_2$ such that $\lambda_{G_1}(x_1) = \lambda_{G_2}(x_2)$

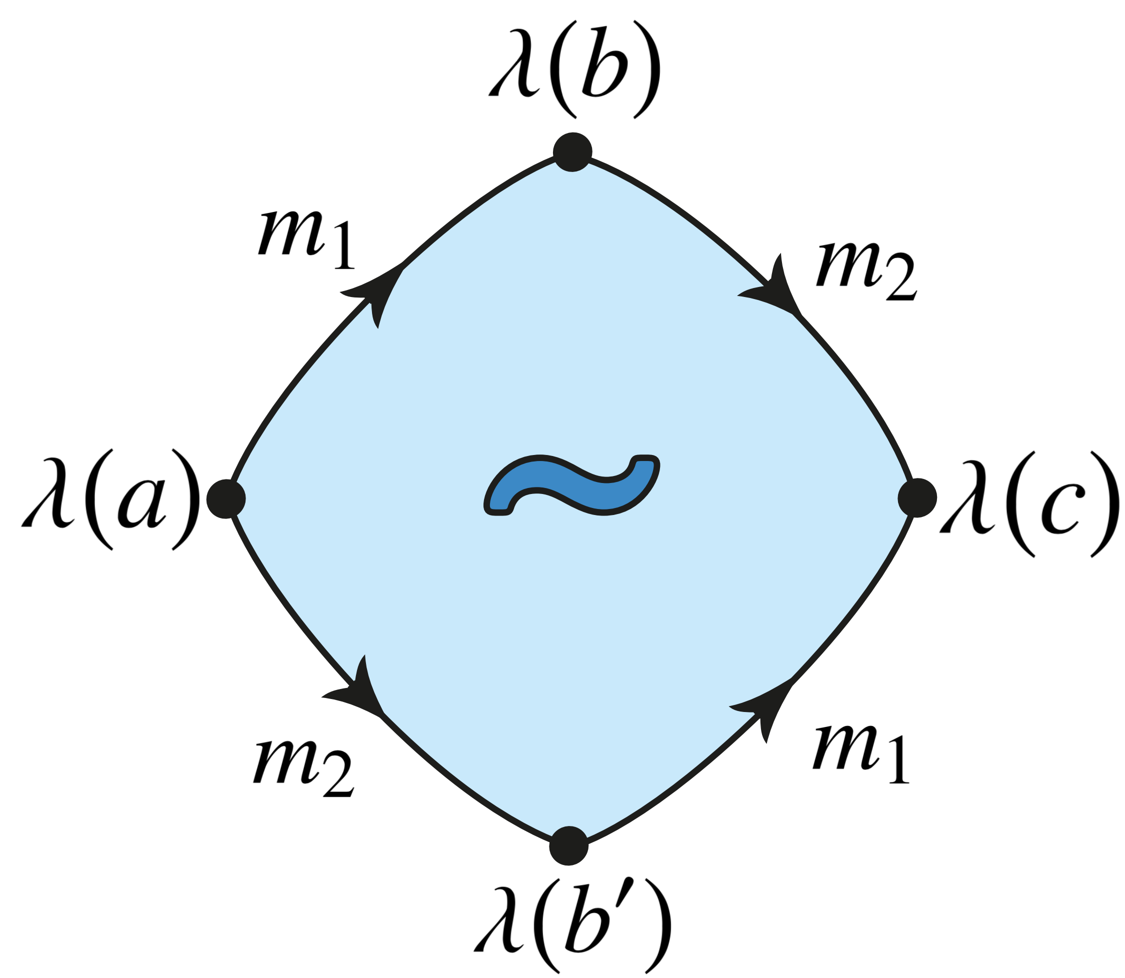
Edges: an edge $x_1 | x_2 \xrightarrow{m \,|\, m} x_1' | x_2'$ is a pair of edges

$$x_1 \xrightarrow{m} x_1' \text{ in } G_1 \quad \text{and} \quad x_2 \xrightarrow{m} x_2' \text{ in } G_2$$

Tiles:



if



Morphism between the two semantics

Morphisms for leaves

$$\begin{array}{ccc} [\![m]\!]_S & \xrightarrow{\qquad\mathscr{L}\qquad} & [\![m]\!]_L \\ (\mu, L) & \longmapsto & L \end{array}$$

They are preserved by the constructions

$$[\![C]\!]_S \xrightarrow{\qquad\mathscr{L}\qquad} [\![C]\!]_L$$

Separation Logic

Hoare triples: $\{P\}C\{Q\}$

P, Q are predicates on *Logical States* $\sigma \in \mathbf{Loc} \rightharpoonup_{fin} (\mathbf{Val} \times \mathbf{Perm})$

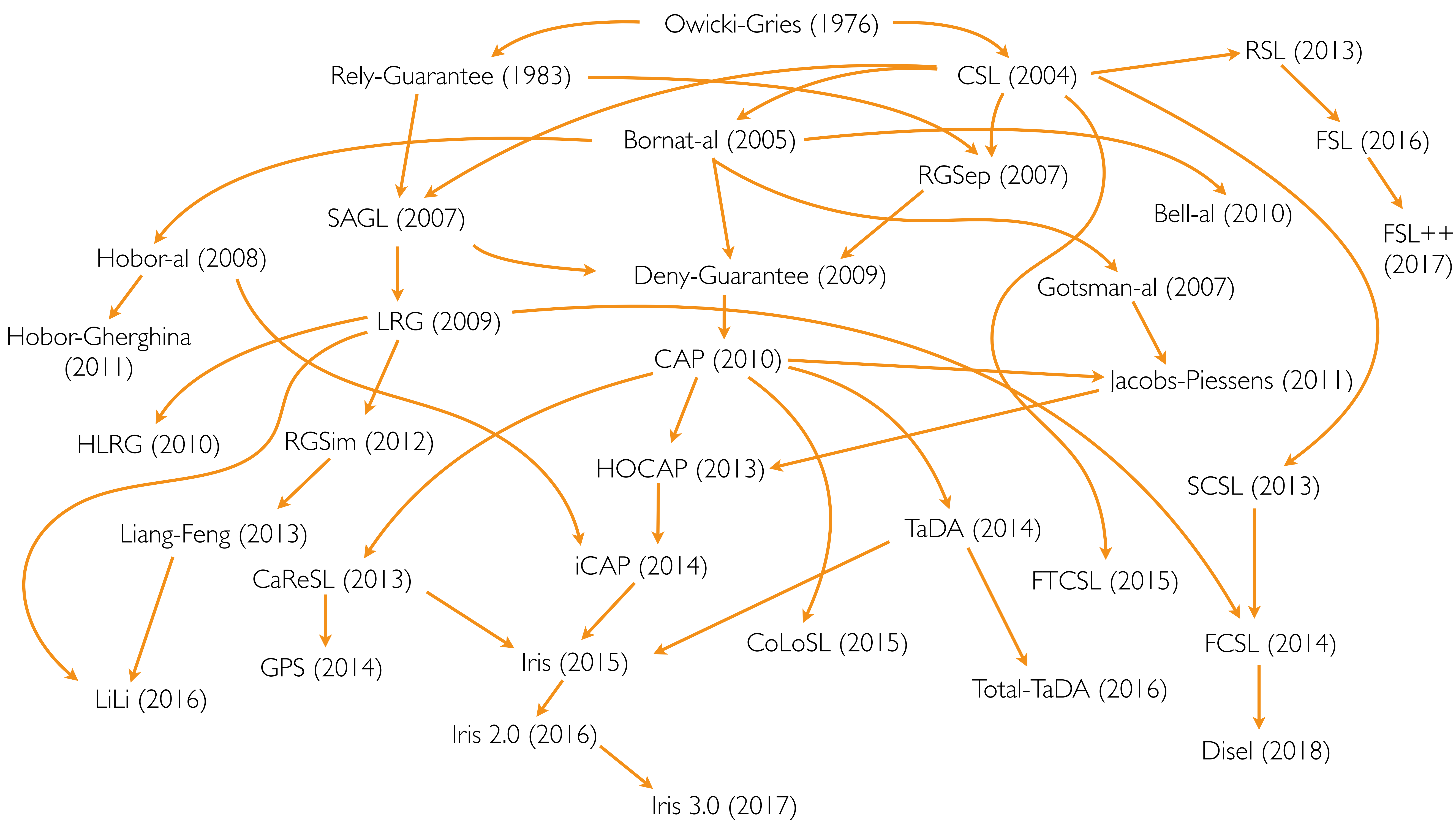
$$P, Q ::= \mathbf{emp} \mid \mathbf{true} \mid \mathbf{false} \mid P \vee Q \mid P \wedge Q \mid P * Q \mid E_1 \mapsto^{p} E_2$$

$$\sigma * \sigma'(a) = \begin{cases} \sigma(a) & \text{if } a \in \mathrm{dom}(\sigma) \setminus \mathrm{dom}(\sigma') \\ \sigma'(a) & \text{if } a \in \mathrm{dom}(\sigma') \setminus \mathrm{dom}(\sigma) \\ (v, p \cdot p') & \text{if } \sigma(a) = (v, p) \text{ and } \sigma'(a) = (v, p') \end{cases}$$

Semantics:

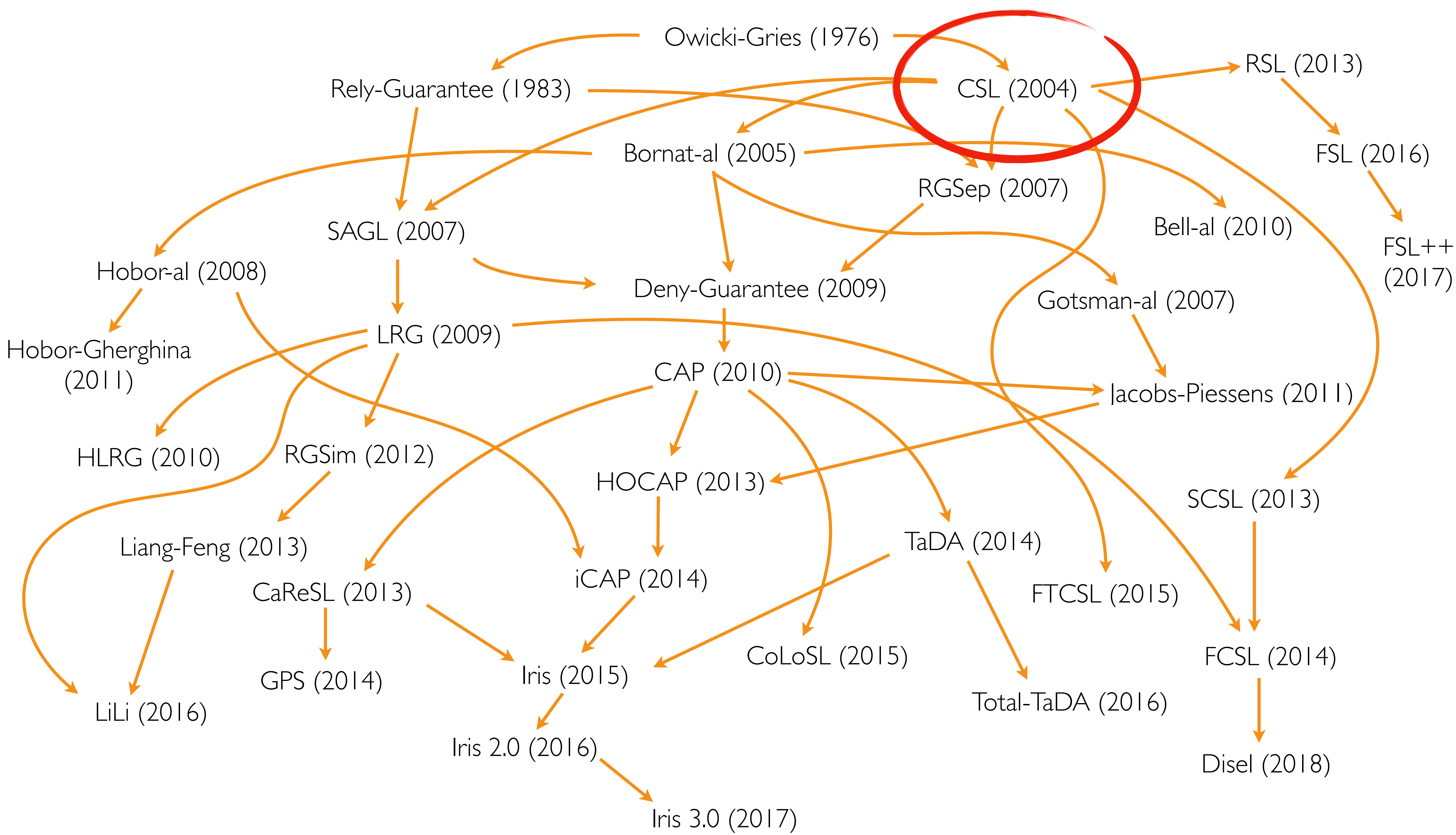
$$\sigma \models P * Q \iff \exists \sigma_1 \sigma_2,\ \sigma = \sigma_1 * \sigma_2 \text{ and } \sigma_1 \models P \text{ and } \sigma_2 \models Q$$

A few concurrent separation logics



by Ilya Sergey

A few concurrent separation logics



by Ilya Sergey

Concurrent Separation Logic (CSL)

$$r_1 : I_1, r_2 : I_2 \vdash \{P\}C\{Q\}$$

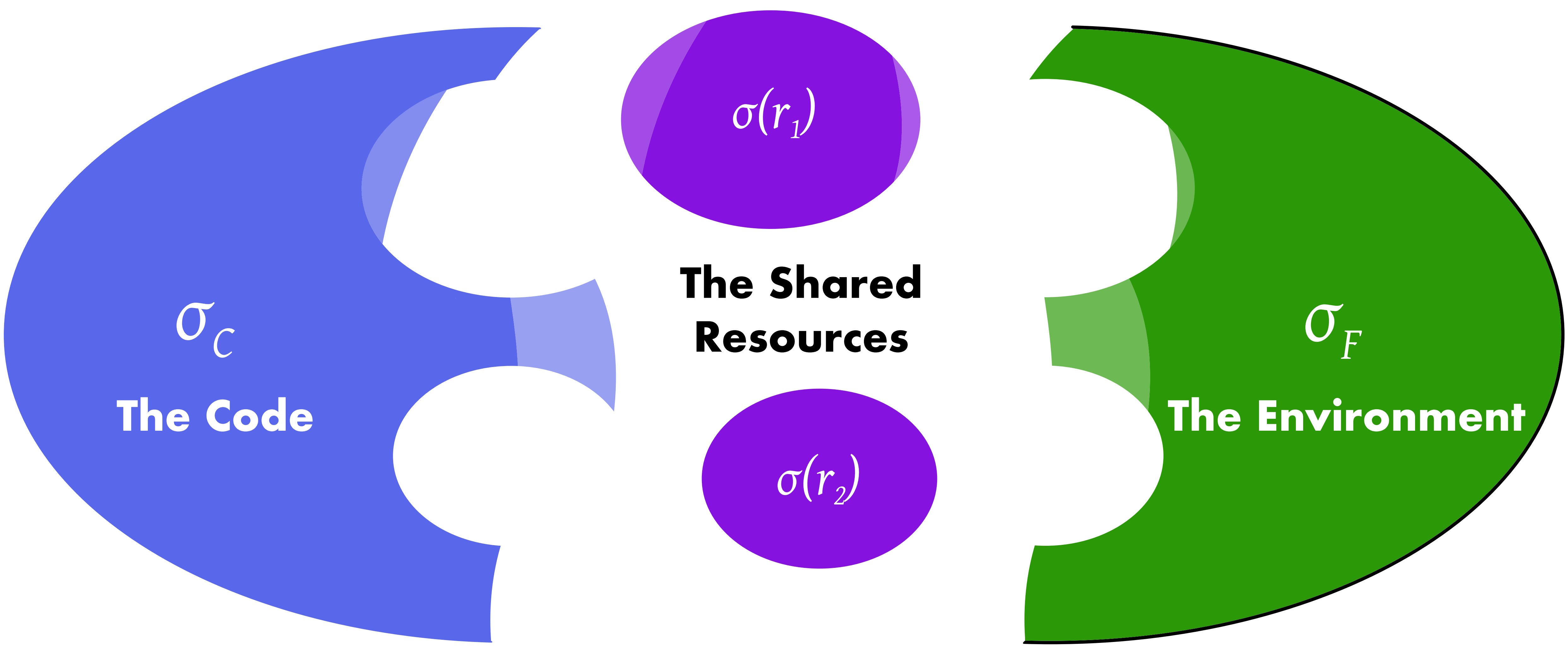
$$\frac{\Gamma \vdash \{P_1\}C_1\{Q_1\} \qquad \Gamma \vdash \{P_2\}C_2\{Q_2\}}{\Gamma \vdash \{P_1 * P_2\}C_1 \parallel C_2\{Q_1 * Q_2\}}\ \textsc{Par}$$

$$\frac{P \Rightarrow \mathrm{def}(B) \qquad \Gamma \vdash \{(P * J) \wedge B\}C\{Q * J\}}{\Gamma, r : J \vdash \{P\}\mathtt{with}\ r\ \mathsf{when}\ B\ \mathsf{do}\ C\{Q\}}\ \textsc{With}$$

$$\frac{\Gamma, r : J \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P * J\}\mathtt{resource}\ r\ \mathsf{do}\ C\{Q * J\}}\ \textsc{Res}$$

Separated States

$$(\sigma_C, \boldsymbol{\sigma}, \sigma_F)$$
$$\Cap$$
$$\textbf{LState} \times (\textbf{LockName} \rightarrow \textbf{LState} + \{C, F\}) \times \textbf{LState}$$

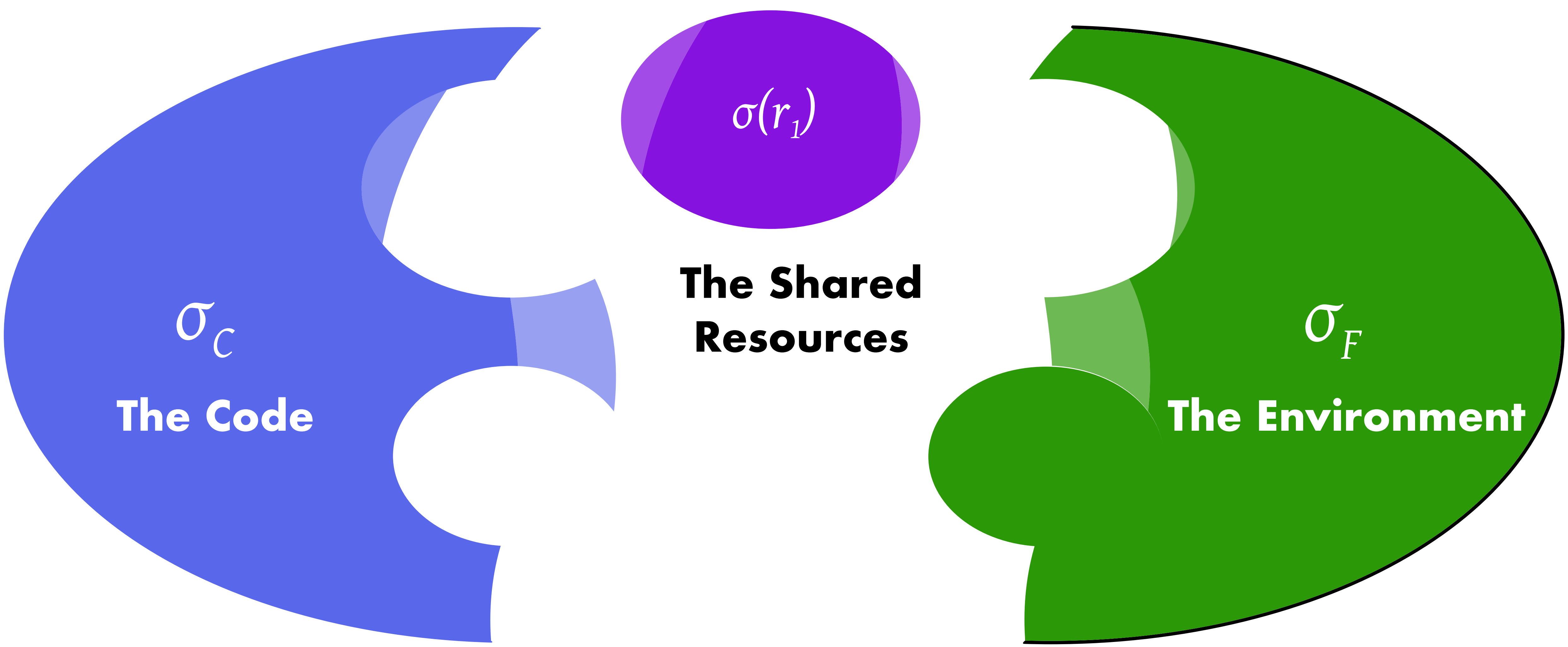


Separated States

$$\left(\sigma_C,\ \boldsymbol{\sigma},\ \sigma_F\right)$$

$$\Cap$$

$$\textbf{LState} \times (\textbf{LockName} \rightarrow \textbf{LState} + \{C, F\}) \times \textbf{LState}$$

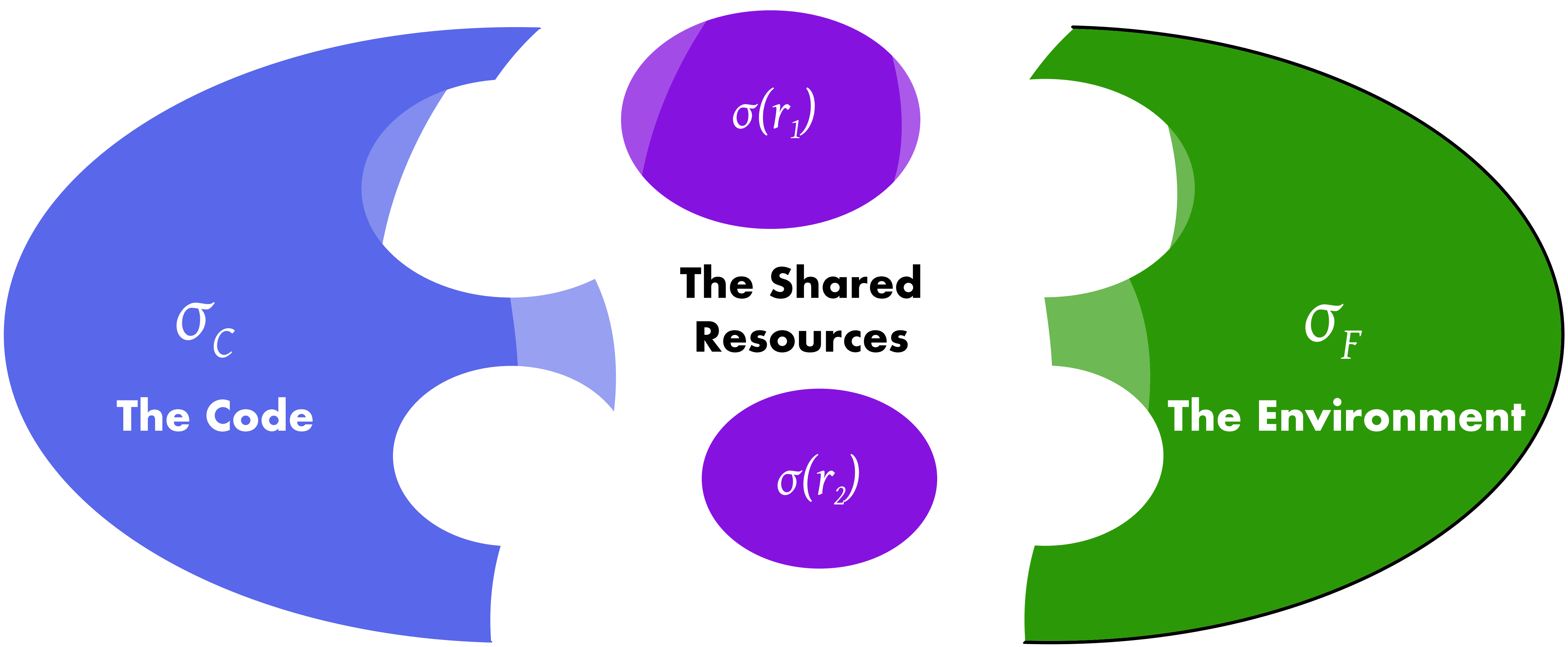


Separated States

$$(\sigma_C, \boldsymbol{\sigma}, \sigma_F)$$

$$\Cap$$

$$\mathbf{LState} \times (\mathbf{LockName} \rightarrow \mathbf{LState} + \{C, F\}) \times \mathbf{LState}$$



Semantics of derivation trees

The semantics of a derivation tree

$$\begin{array}{c} \vdots\ \pi \\ \Gamma \vdash \{P\}C\{Q\} \end{array}$$

is an Asynchronous Transition System over **separated states**

$$\lambda : [\![\pi]\!]_{Sep} \longrightarrow \curlywedge_{Sep}$$

- The initial states are all the separated states that satisfy P
- The final states all satisfy Q
- Each of the $\boldsymbol{\sigma}$ satisfies Γ

Machine Model of Separated States

$$\text{大}_{Sep}$$

States: separated states $(\sigma_C, \boldsymbol{\sigma}, \sigma_F)$

Two kinds of transitions:

$$(\sigma_C, \boldsymbol{\sigma}, \sigma_F) \xrightarrow{m} (\sigma'_C, \boldsymbol{\sigma}', \sigma_F) \qquad (\sigma_C, \boldsymbol{\sigma}, \sigma_F) \xrightarrow{\color{red}{m}} (\sigma_C, \boldsymbol{\sigma}', \sigma'_F)$$

Tiles: so that they correspond to tiles in 大_S

Asynchronous Graph Morphism

There is an asynchronous graph morphism

$$\left[\!\left[\frac{\vdots\, \pi}{\Gamma \vdash \{P\} C \{Q\}} \right]\!\right]_{Sep} \xrightarrow{\mathscr{S}} [\![C]\!]_S$$

with

$$(\sigma_C, \boldsymbol{\sigma}, \sigma_F) \quad \longmapsto \quad \sigma_C * \left\{ \mathop{\circledast}_{r \in \mathrm{dom}(\boldsymbol{\sigma})} \boldsymbol{\sigma}(r) \right\} * \sigma_F$$

Soundness theorems

- “A well specified program does not go wrong”
  - Memory safety, etc…
  - Data-race freedom
- Precondition, postcondition

1-soundness

Theorem 1

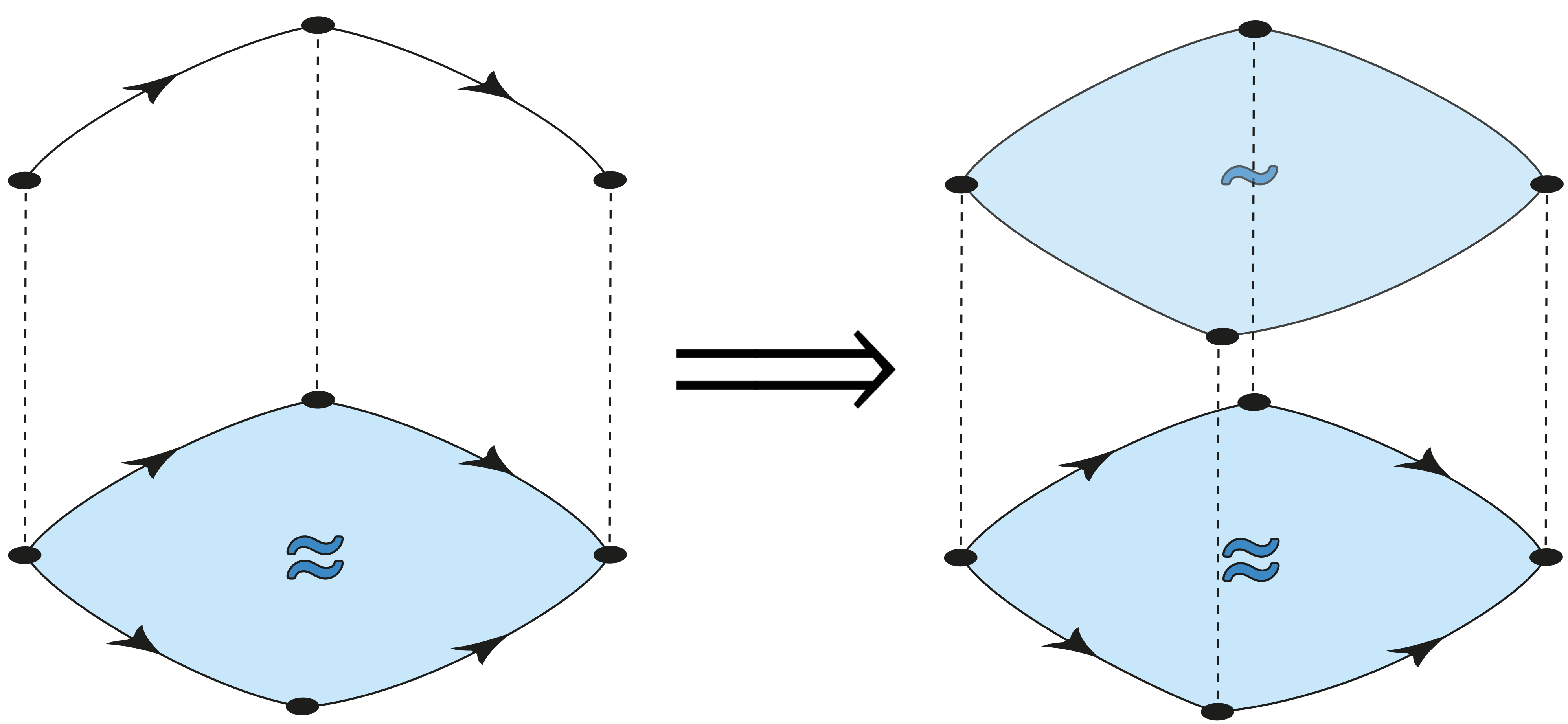
$\mathscr{S}$ is an op-fibration on Code transitions.

$$
\begin{tikzcd}
(\sigma_C, \boldsymbol{\sigma}, \sigma_F) \arrow[d, dotted, dash, "\mathscr{S}"'] & \\
\mathfrak{s}_1 \arrow[r, "m"'] & \mathfrak{s}_2
\end{tikzcd}
\quad \Longrightarrow \quad
\begin{tikzcd}
(\sigma_C, \boldsymbol{\sigma}, \sigma_F) \arrow[r, "m"'] \arrow[d, dotted, dash, "\mathscr{S}"'] & (\sigma'_C, \boldsymbol{\sigma}', \sigma_F) \arrow[d, dotted, dash, "\mathscr{S}"] \\
\mathfrak{s}_1 \arrow[r, "m"'] & \mathfrak{s}_2
\end{tikzcd}
$$

2-soundness

Theorem 2

$$\left[\!\!\left[\dfrac{\vdots\, \pi}{\Gamma \vdash \{P\}C\{Q\}} \right]\!\!\right]_{Sep} \xrightarrow{\quad \mathscr{L} \circ \mathscr{S} \quad} [\![C]\!]_L$$ is a 2-fibration.



The End