Secure information flow for synchronous reactive programs

Ilaria Castellani

INRIA Sophia Antipolis

OPCT Workshop Bertinoro, 18-21 June, 2014

[Based on TGC'13 talk, joint work with Pejman Attar]

Synopsis

- Motivation
- Synchronous reactive model
- Syntax of CRL (Core Reactive Language)
- Semantics of CRL and properties
- Fine-grained and coarse-grained bisimilarity
- Secure information flow: f-grained and c-grained reactive noninterference (RNI)
- Security type system
- Related work and open questions

Problem and motivation

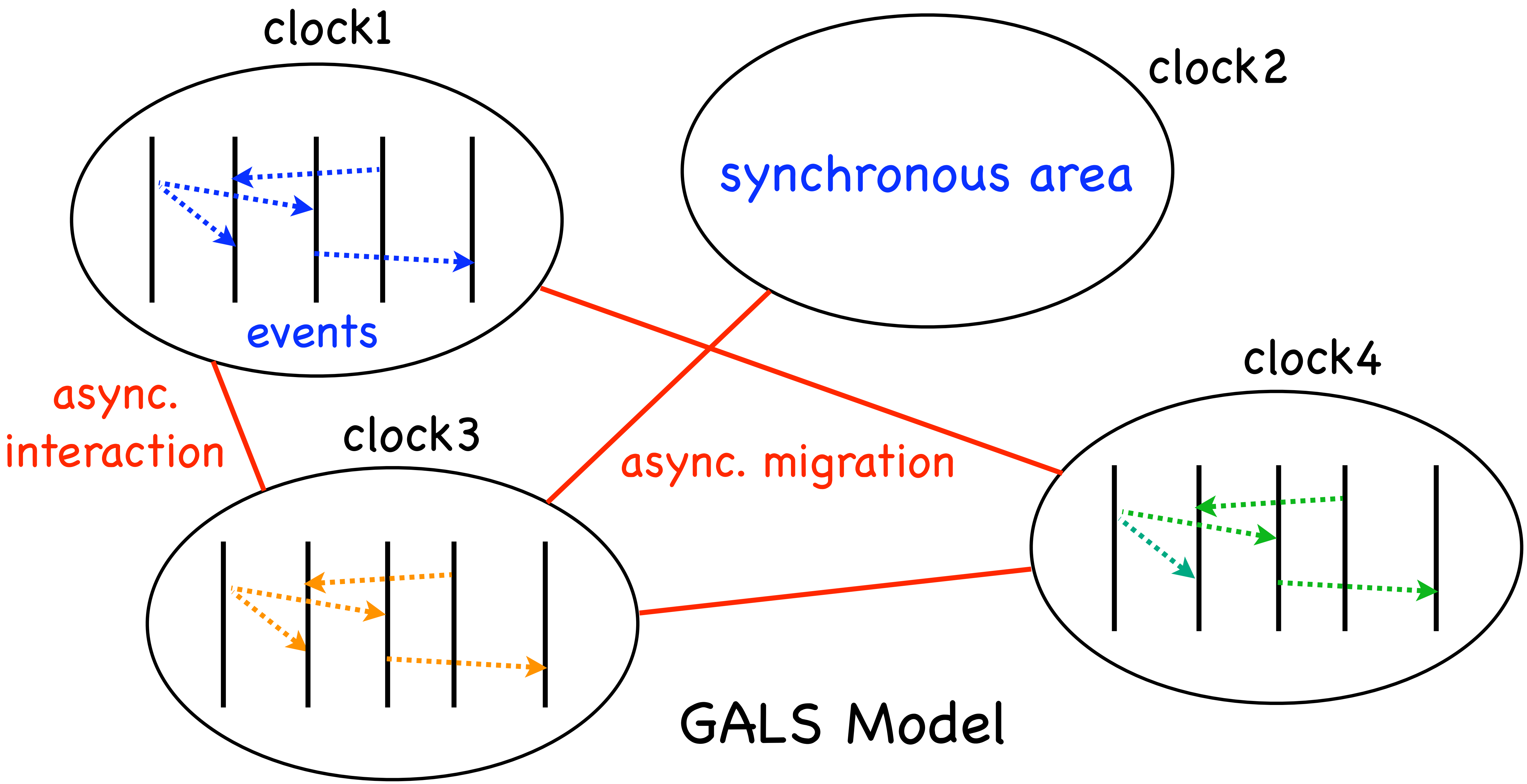
Current systems (e.g., web browsers) are often reactive: they listen and react to the environment by means of events

Mutually distrusting parties need confidentiality guarantees for their data

-> Goal: ensure secure information flow (end-to-end protection of data confidentiality) in reactive systems

Synchronous Reactive Model

Synchronous areas within a GALS architecture
(GALS = Globally Asynchronous, Locally Synchronous).



Synchronous Languages

Cooperative parallelism + broadcast events

instant = period of time during which all threads compute up to termination or suspension (suspension = control yield or waiting for an event)

Reactive variant of ESTEREL [Berry et al., mid 80's]:

-> SL (Synchronous Language) [Boussinot, 1996]

Delayed reaction to absence of events => no causality cycles, monotonic computations

Synchronous parallelism

Asymmetric parallel operator $s \nmid s'$

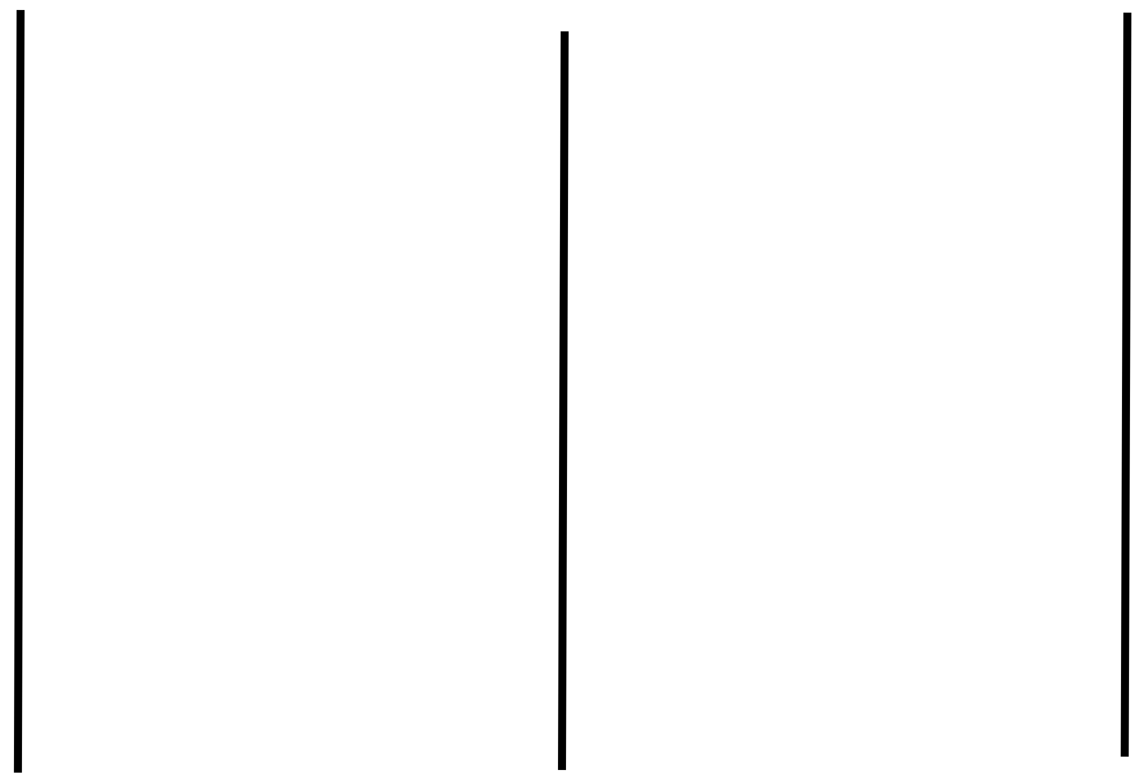
Priority to the left

Synchronous parallelism

Asymmetric parallel operator $s \nmid s'$

$$s_1 \quad \nmid \quad s_2 \quad \nmid \quad s_3$$

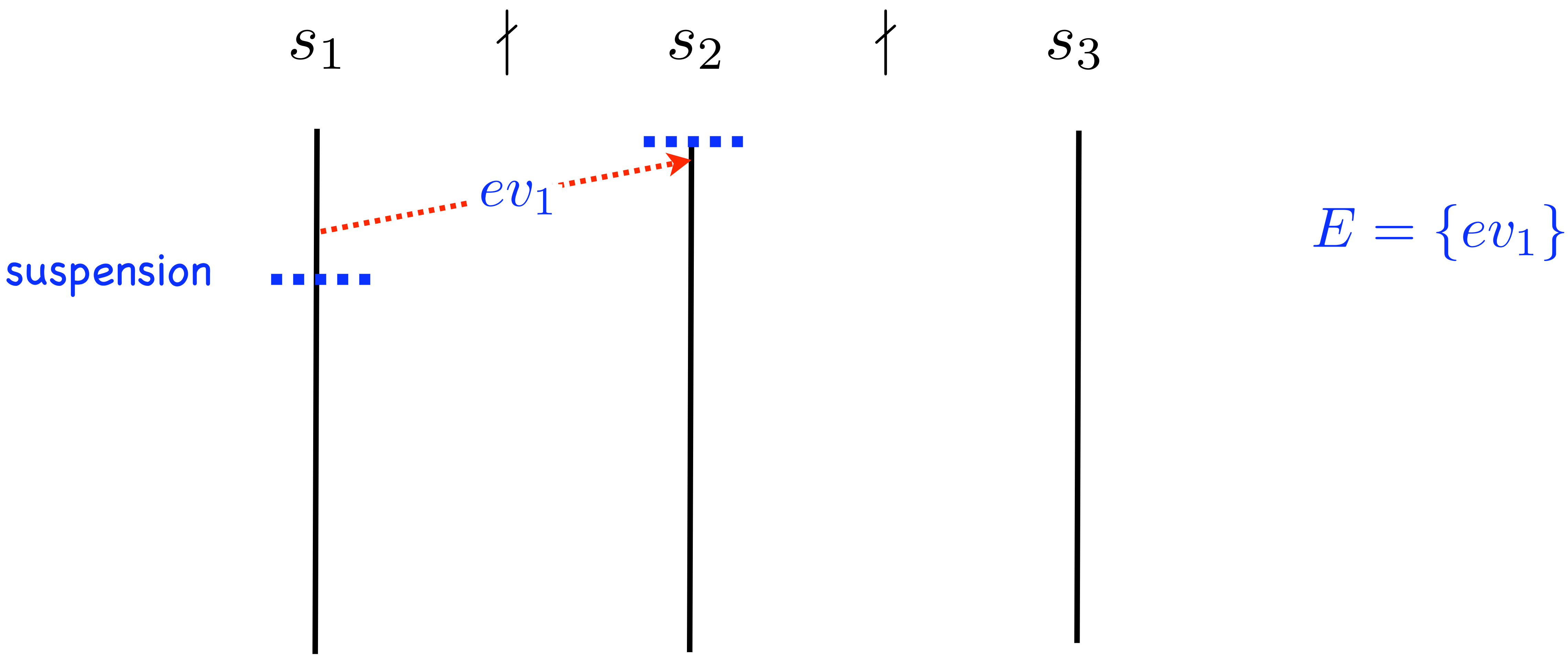
$E = \emptyset$



Programs are executed in an event environment E

Synchronous parallelism

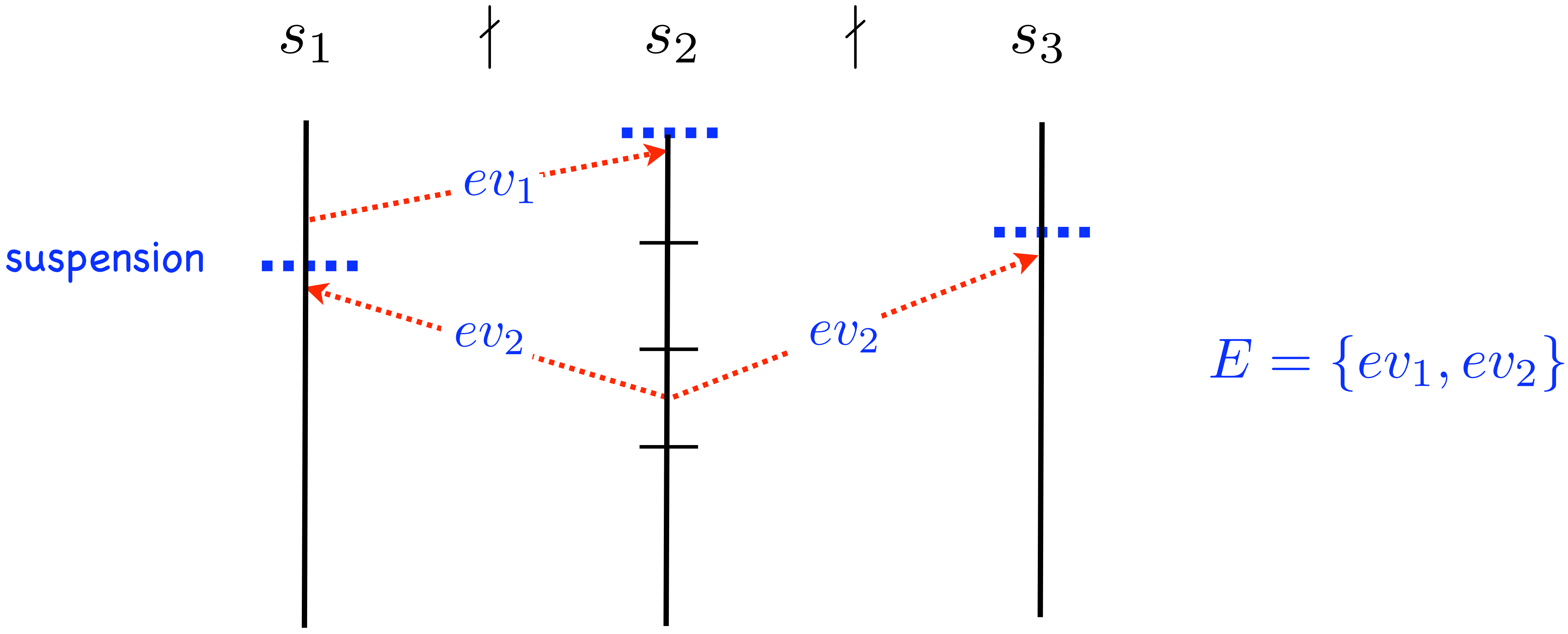
Asymmetric parallel operator $s \nmid s'$



s_1 executes first, generating ev_1

Synchronous parallelism

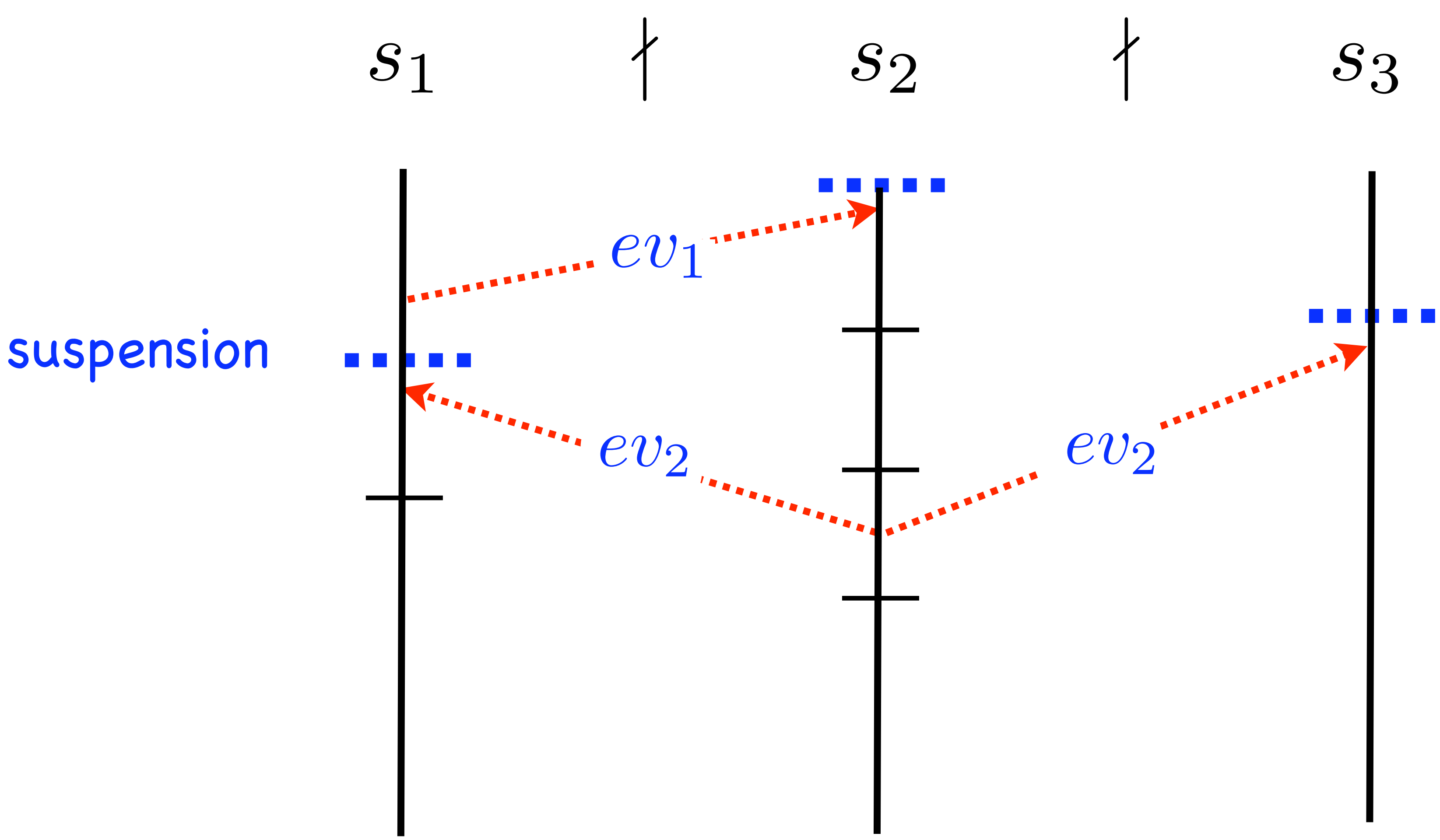
Asymmetric parallel operator $s \nmid s'$



s_1 suspends, s_2 gets the control and generates ev_2

Synchronous parallelism

Asymmetric parallel operator $s \nmid s'$

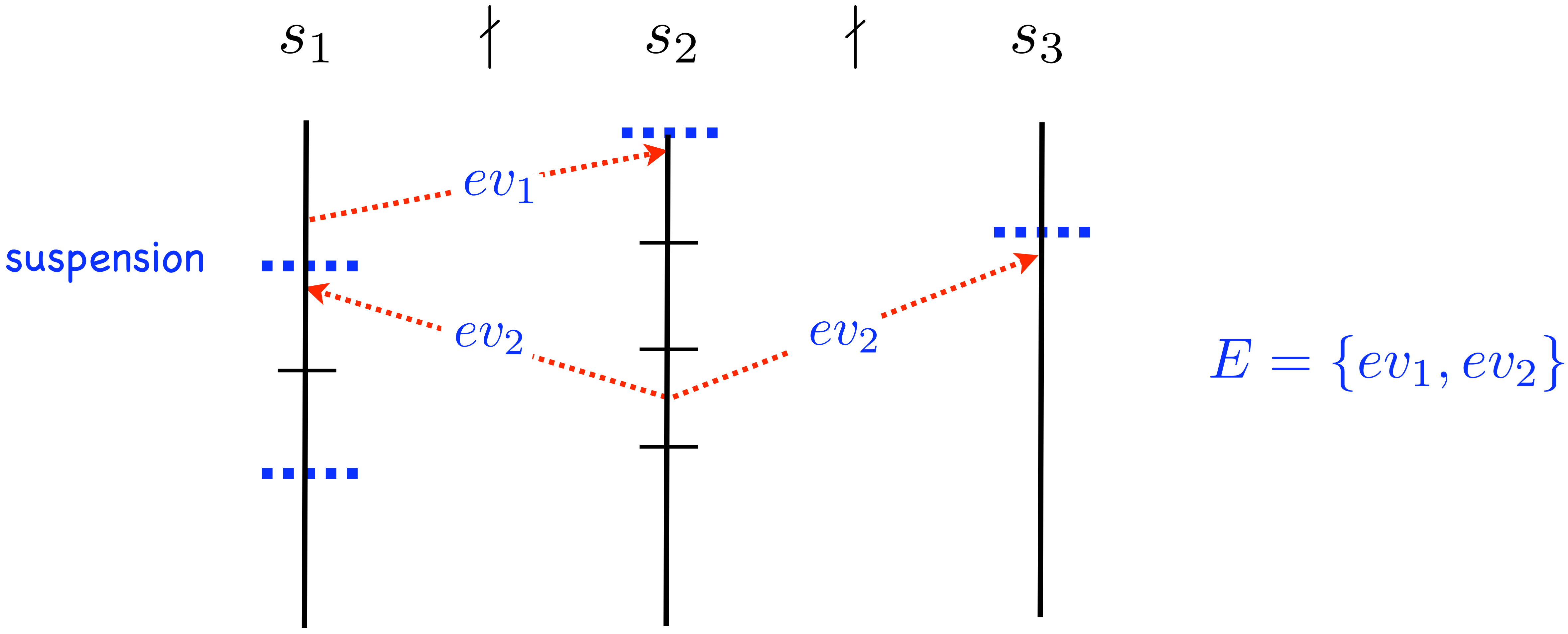


$E = \{ev_1, ev_2\}$

s_1 unblocks and gets back the control

Synchronous parallelism

Asymmetric parallel operator $s \nmid s'$

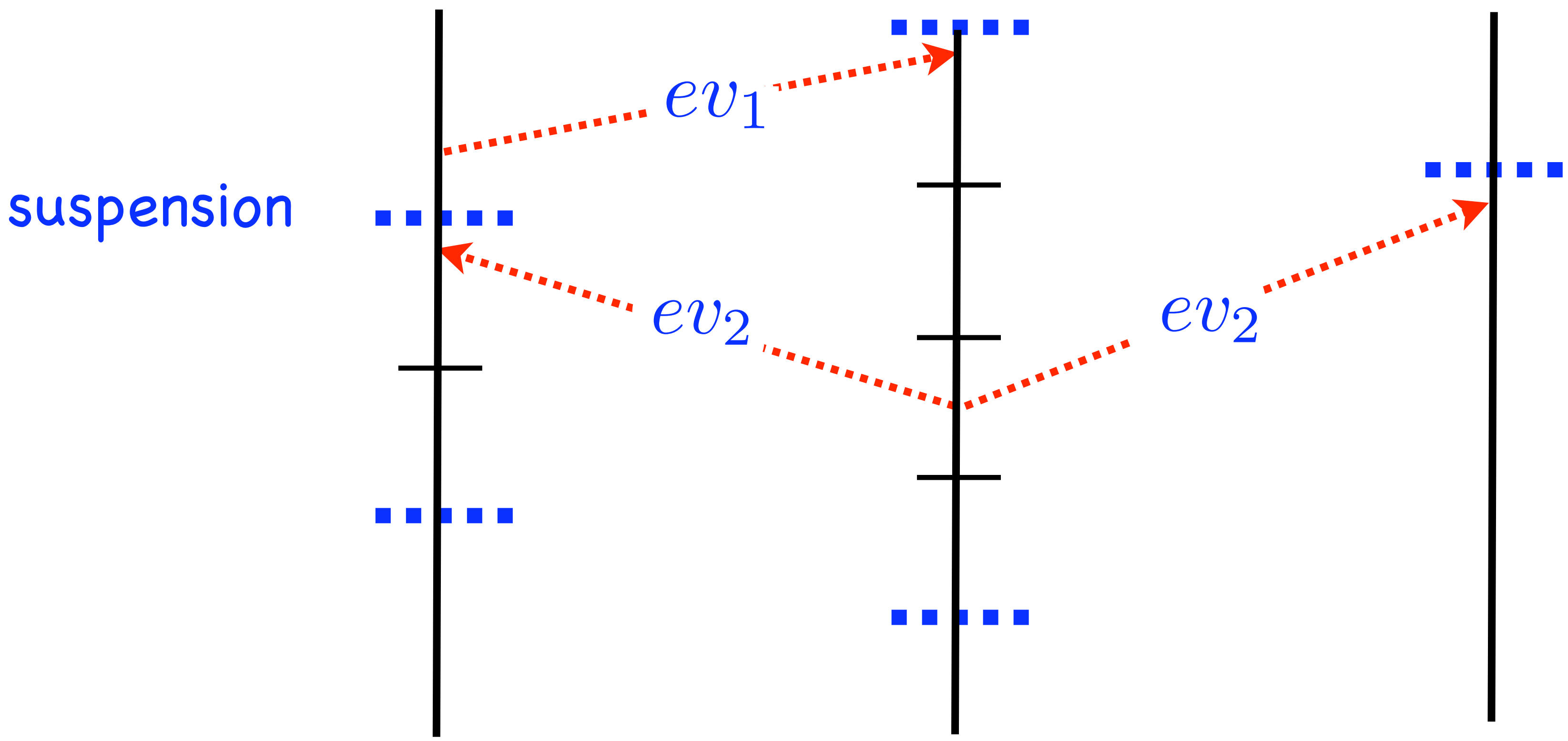


s_1 suspends again, s_2 gets the control

Synchronous parallelism

Asymmetric parallel operator $s \nmid s'$

$$s_1 \quad \nmid \quad s_2 \quad \nmid \quad s_3$$

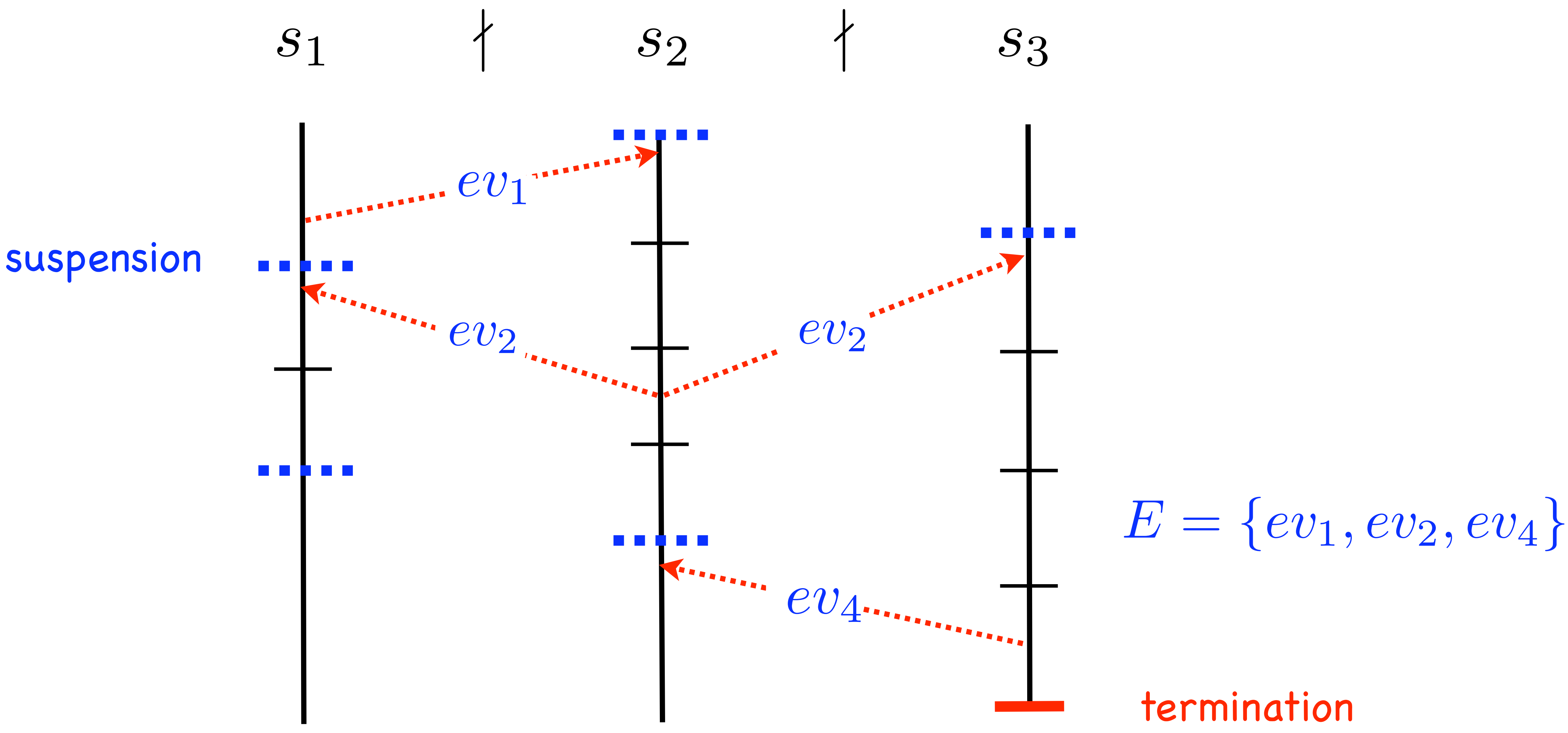


$E = \{ev_1, ev_2\}$

both s_1 and s_2 are suspended, s_3 gets the control

Synchronous parallelism

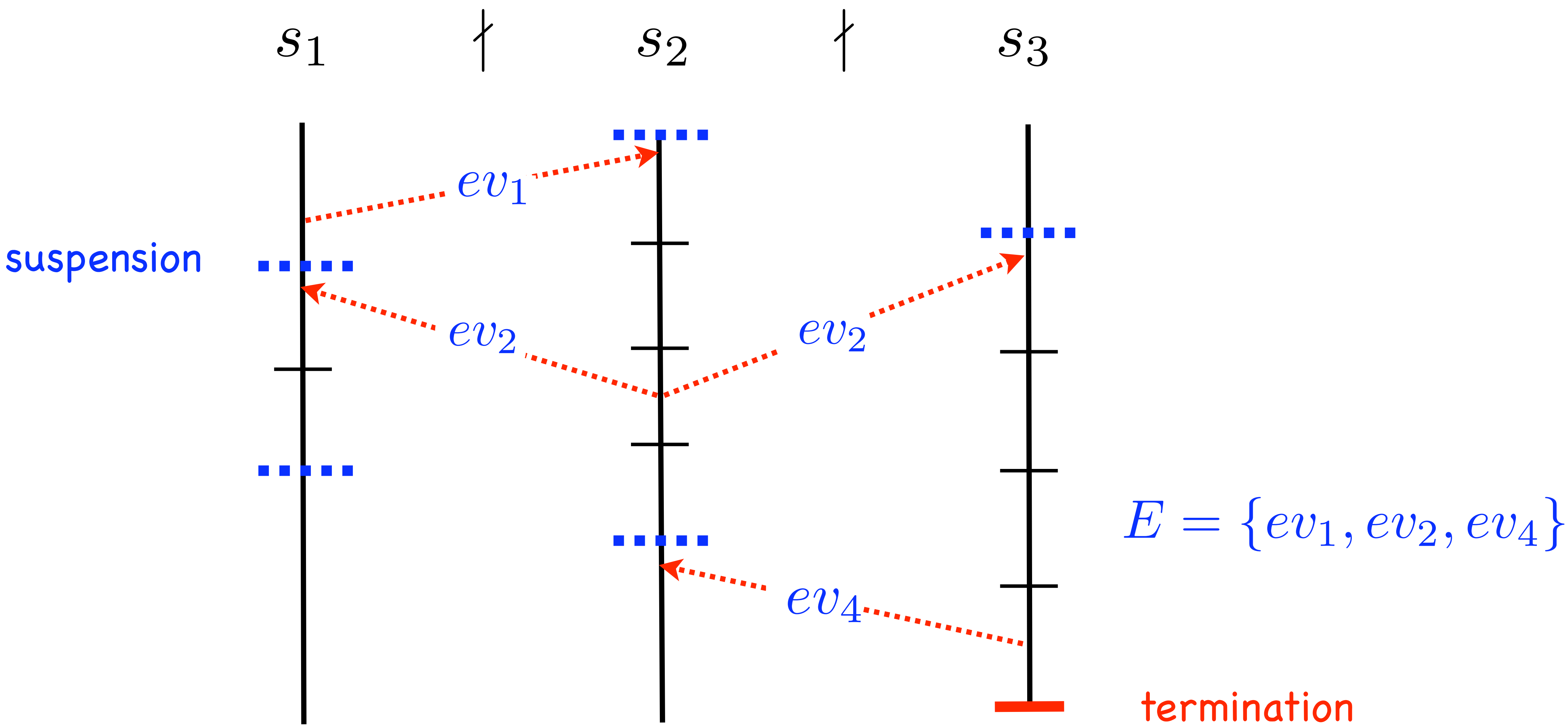
Asymmetric parallel operator $s \nmid s'$



s_3 executes till termination, generating ev_4

Synchronous parallelism

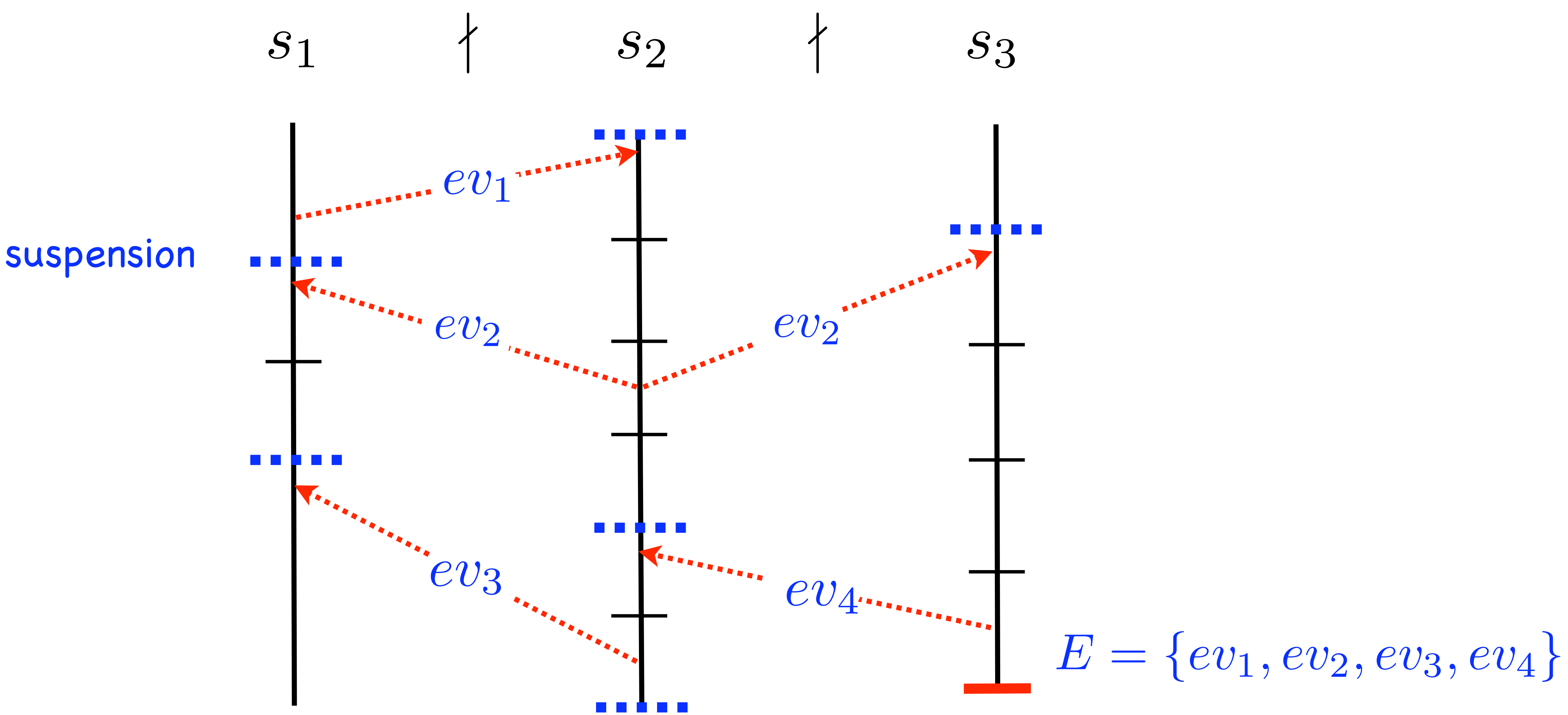
Asymmetric parallel operator $s \nmid s'$



the control goes back to s_2

Synchronous parallelism

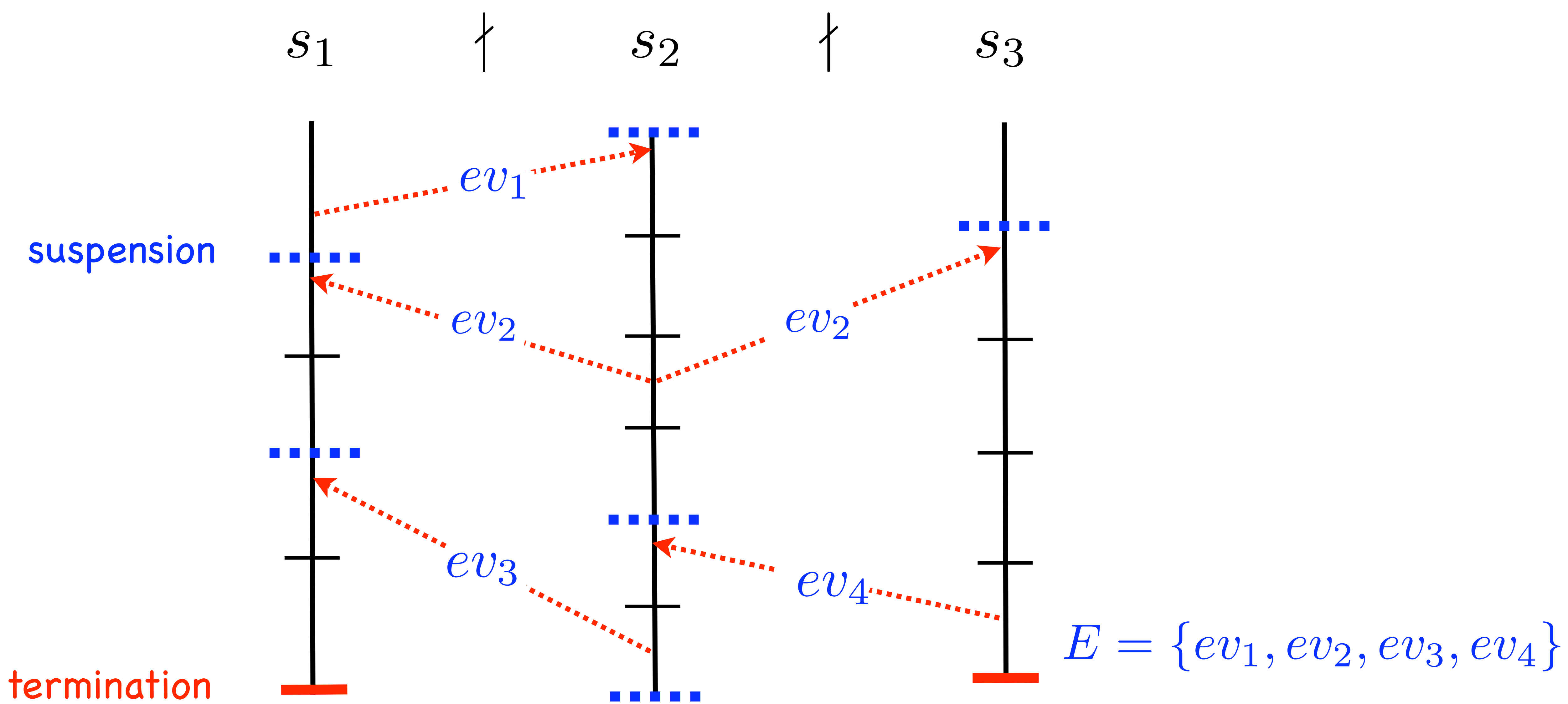
Asymmetric parallel operator $s \nmid s'$



the control goes back to s_1

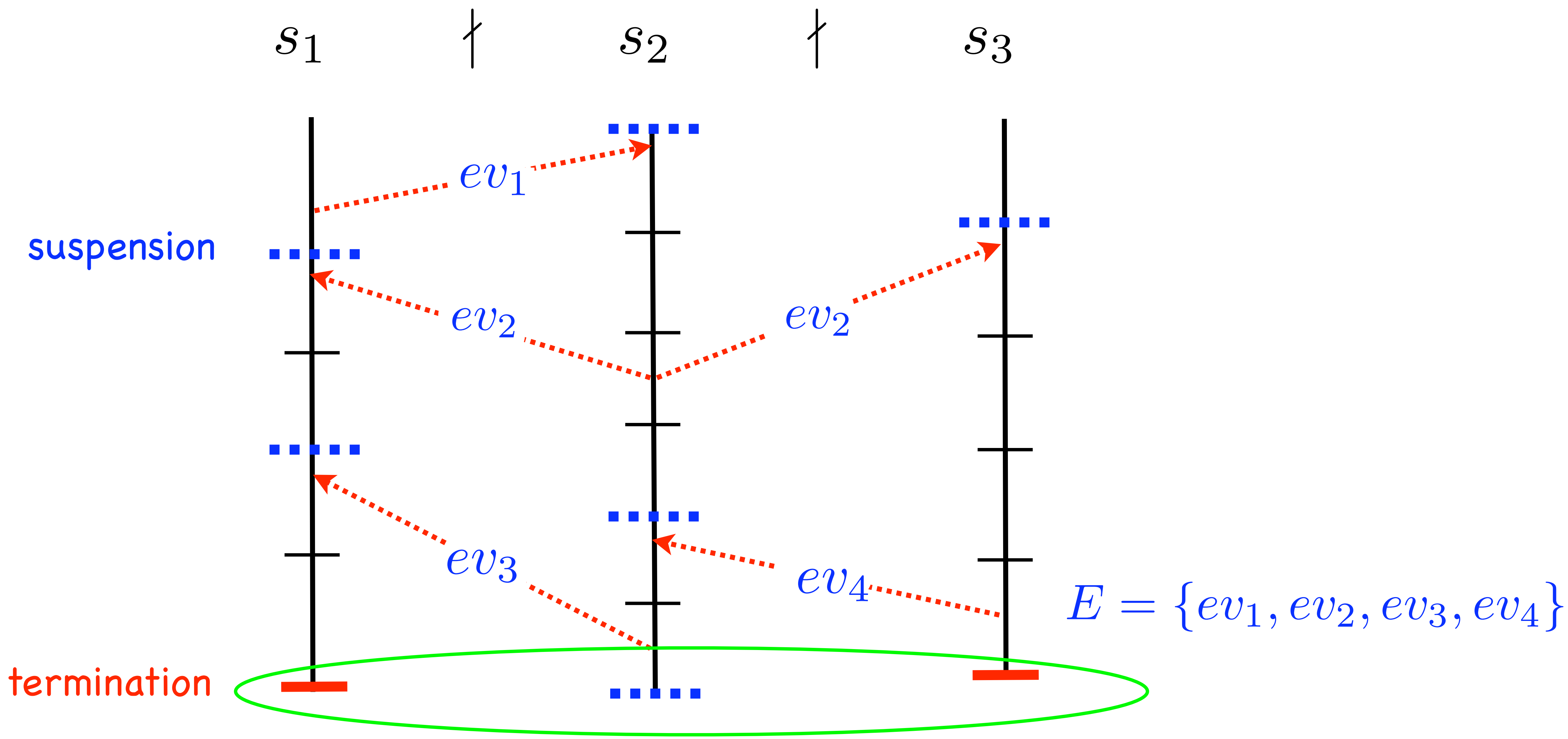
Synchronous parallelism

Asymmetric parallel operator $s \nmid s'$



End of instant

Asymmetric parallel operator $s \nmid s'$



Synchronisation barrier

Syntax of CRL

Expressions

$$exp ::= v \;\mid\; x \;\mid\; f(\overrightarrow{exp})$$

Programs

$$\begin{aligned} s ::= \quad & \texttt{nothing} \mid (\texttt{if}\; exp\; \texttt{then}\; s\; \texttt{else}\; s) \mid s; s \mid (s \nmid s) \mid \\ & \texttt{cooperate} \mid \texttt{generate}\; ev \mid \texttt{await}\; ev \mid \texttt{do}\; s\; \texttt{watching}\; ev \mid \\ & (\texttt{loop}\; s) \mid (\texttt{repeat}\; exp\; \texttt{do}\; s) \end{aligned}$$

Syntax of CRL

Expressions

$$exp ::= v \mid x \mid f(\overrightarrow{exp})$$

Programs

$$s ::= \texttt{nothing} \mid (\texttt{if}\ exp\ \texttt{then}\ s\ \texttt{else}\ s) \mid s; s \mid (s \nmid s) \mid$$

$$\texttt{cooperate} \mid \texttt{generate}\ ev \mid \texttt{await}\ ev \mid \texttt{do}\ s\ \texttt{watching}\ ev \mid$$

$$(\texttt{loop}\ s) \mid (\texttt{repeat}\ exp\ \texttt{do}\ s)$$

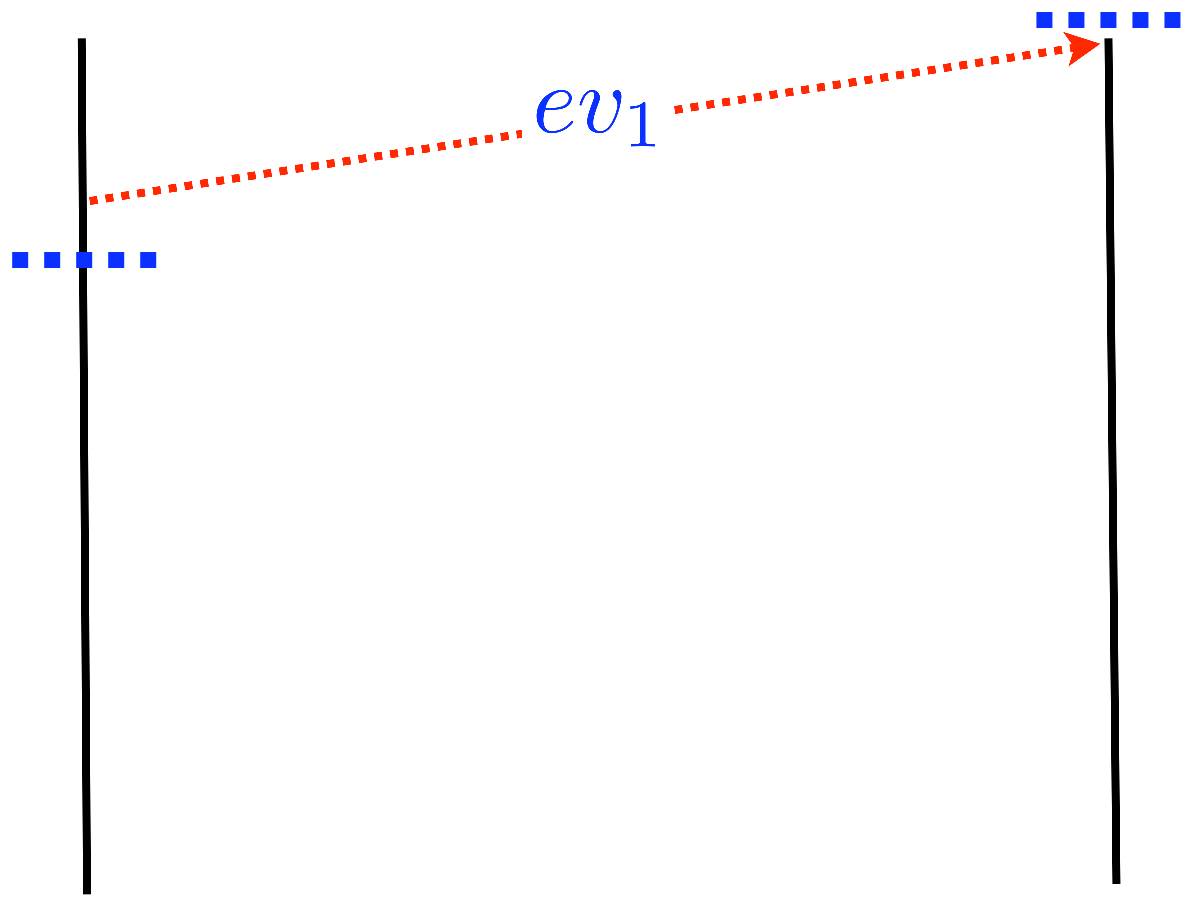
Reactive constructs

$$s_1 = \texttt{generate}\, ev_1; \texttt{await}\, ev_2; \texttt{cooperate}; \texttt{generate}\, ev_3 \quad \nmid \quad s_2 = \texttt{await}\, ev_1; \texttt{generate}\, ev_2; \texttt{await}\, ev_3;$$

$$E = \emptyset$$

Reactive constructs

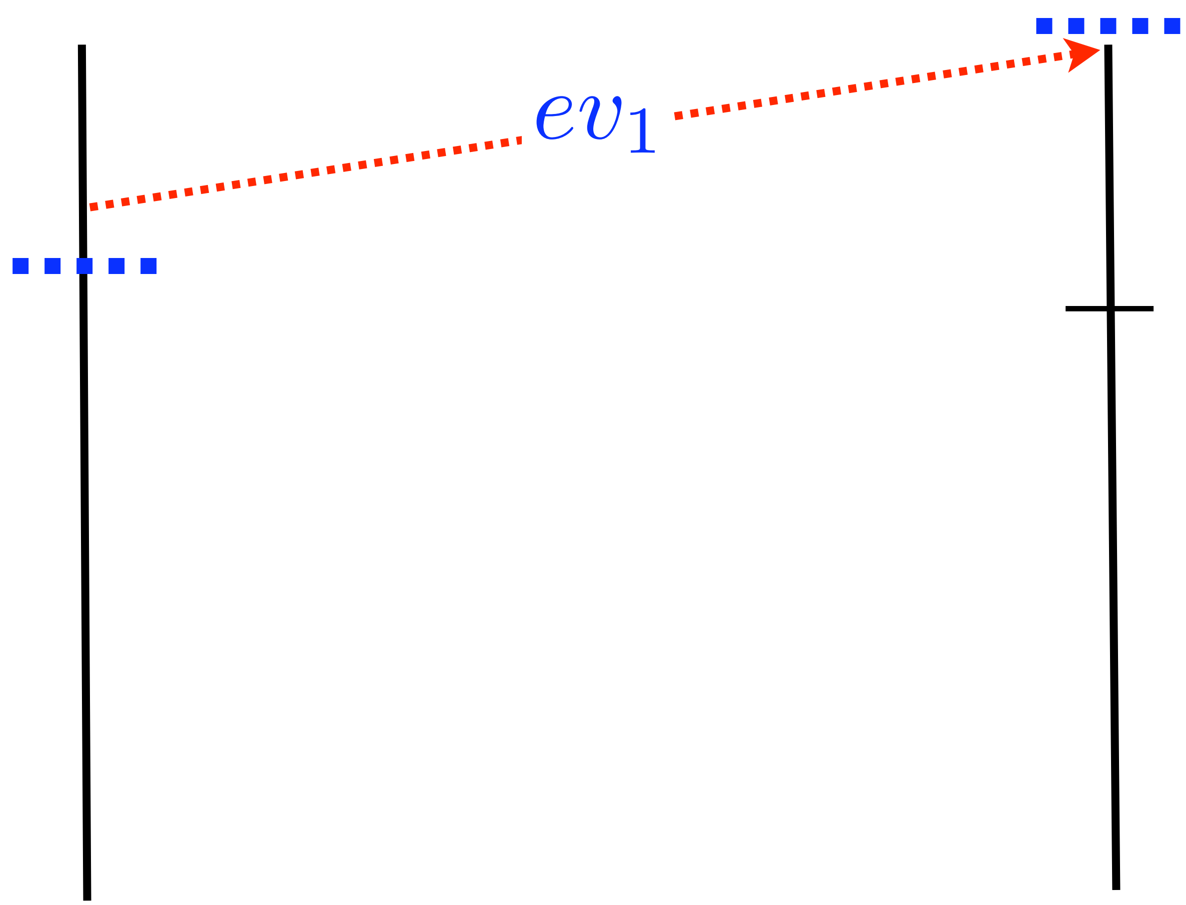
$$s_1 \;=\; \texttt{generate}\; ev_1;\; \texttt{await}\; ev_2;\; \texttt{cooperate};\; \texttt{generate}\; ev_3 \quad \nmid \quad s_2 \;=\; \texttt{await}\; ev_1;\; \texttt{generate}\; ev_2;\; \texttt{await}\; ev_3$$



$E = \{ev_1\}$

Reactive constructs

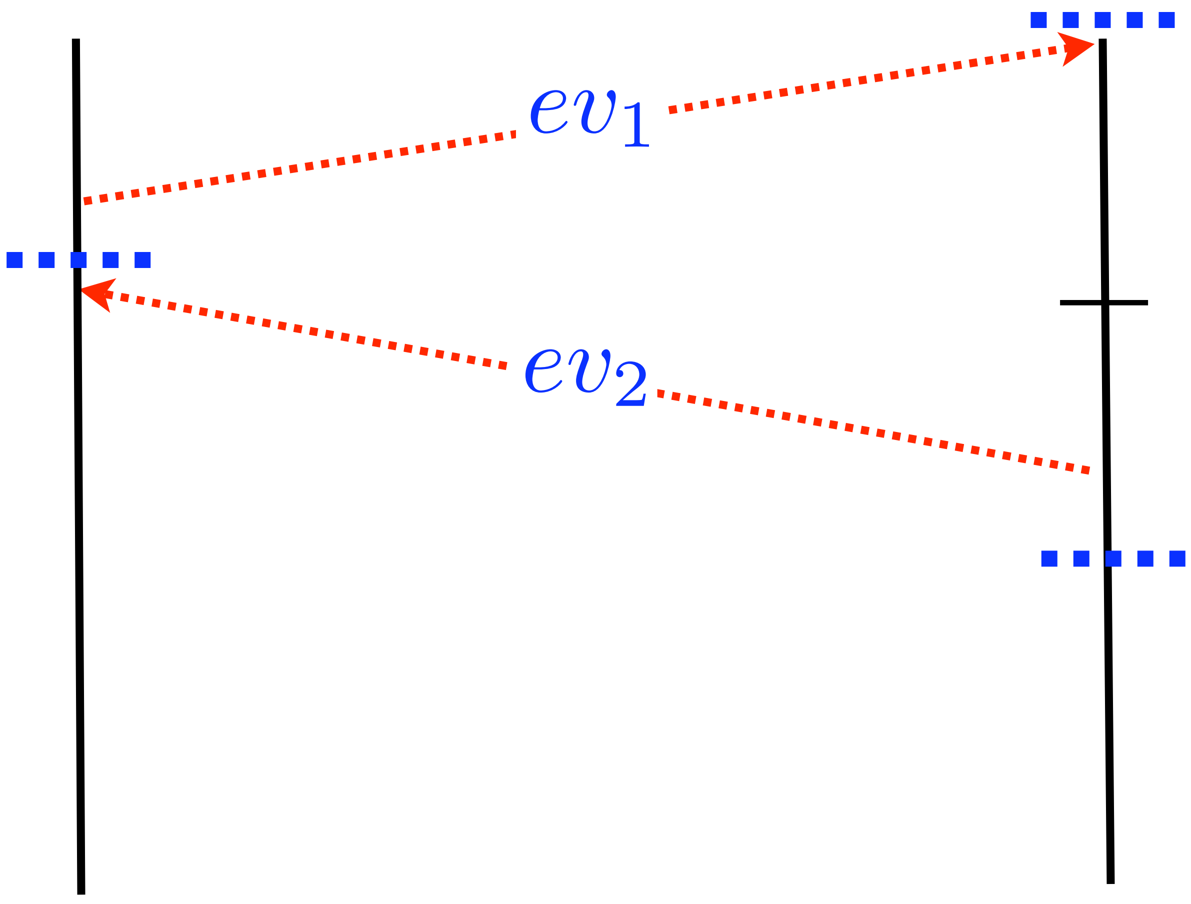
$$
\begin{array}{lll}
s_1 & = & \texttt{generate}\ ev_1; \\
 & & \texttt{await}\ ev_2; \\
 & & \texttt{cooperate}; \\
 & & \texttt{generate}\ ev_3
\end{array}
\quad \not| \quad
\begin{array}{lll}
s_2 & = & \texttt{await}\ ev_1; \\
 & & \texttt{generate}\ ev_2; \\
 & & \texttt{await}\ ev_3
\end{array}
$$



$$E = \{ev_1\}$$

Reactive constructs

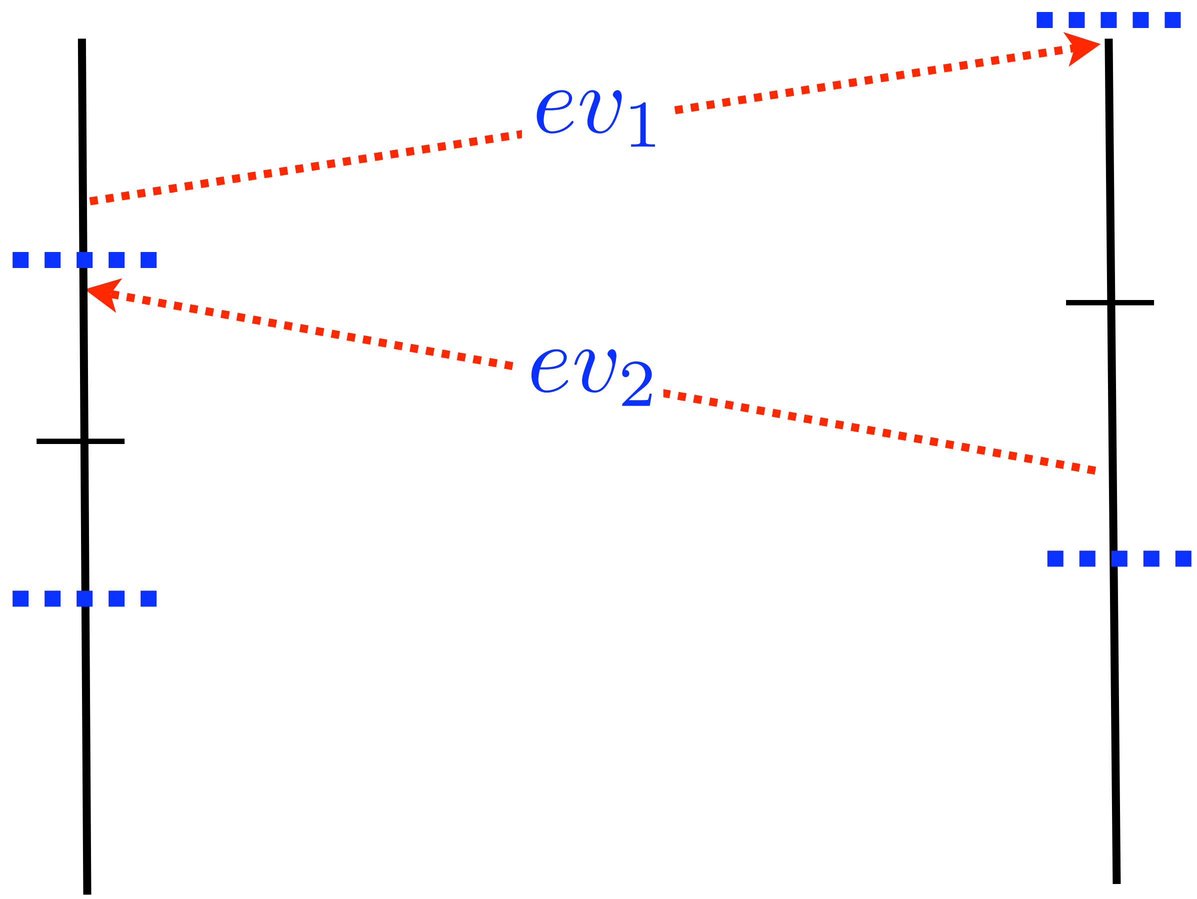
$$s_1 = \texttt{generate}\ ev_1;\ \texttt{await}\ ev_2;\ \texttt{cooperate};\ \texttt{generate}\ ev_3 \quad \nmid \quad s_2 = \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2;\ \texttt{await}\ ev_3$$



$E = \{ev_1, ev_2\}$

Reactive constructs

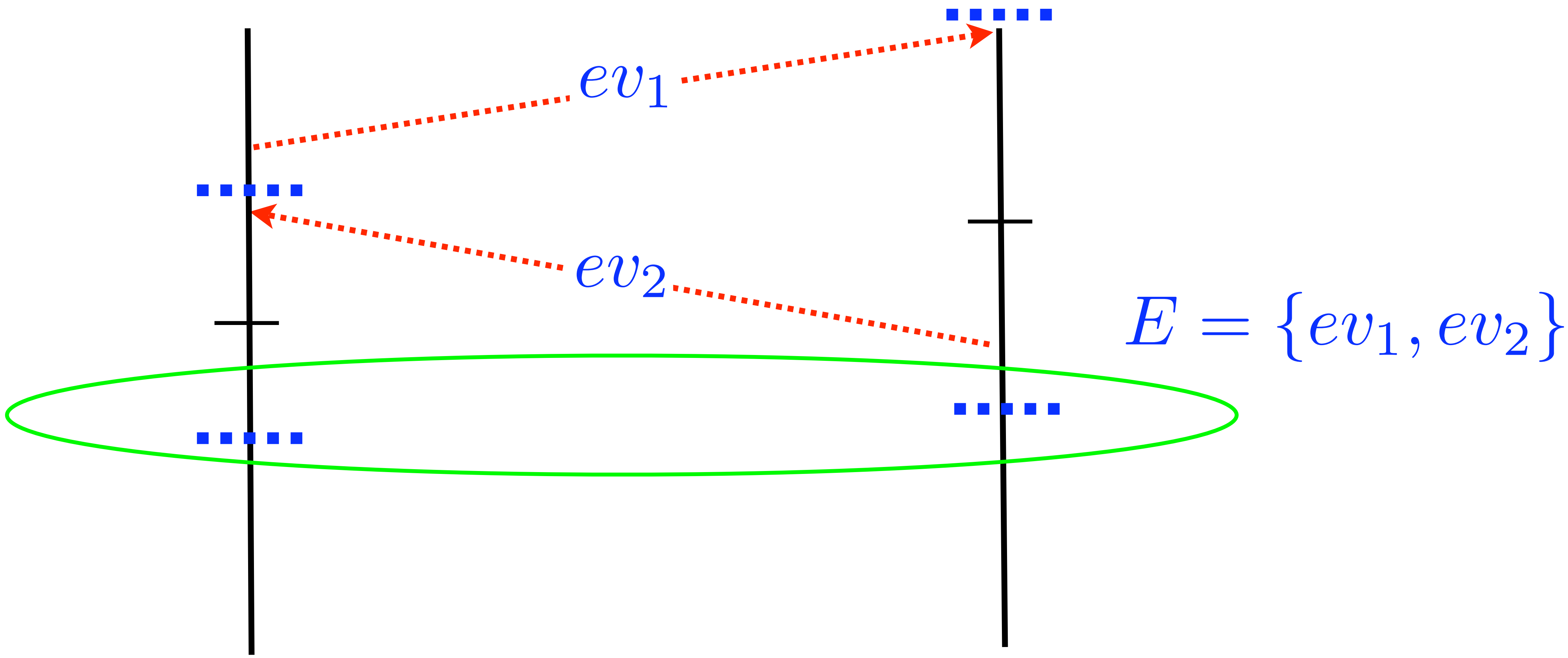
$$s_1 = \texttt{generate}\ ev_1;\ \texttt{await}\ ev_2;\ \texttt{cooperate};\ \texttt{generate}\ ev_3 \quad \nmid \quad s_2 = \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2;\ \texttt{await}\ ev_3$$



$$E = \{ev_1, ev_2\}$$

End of instant

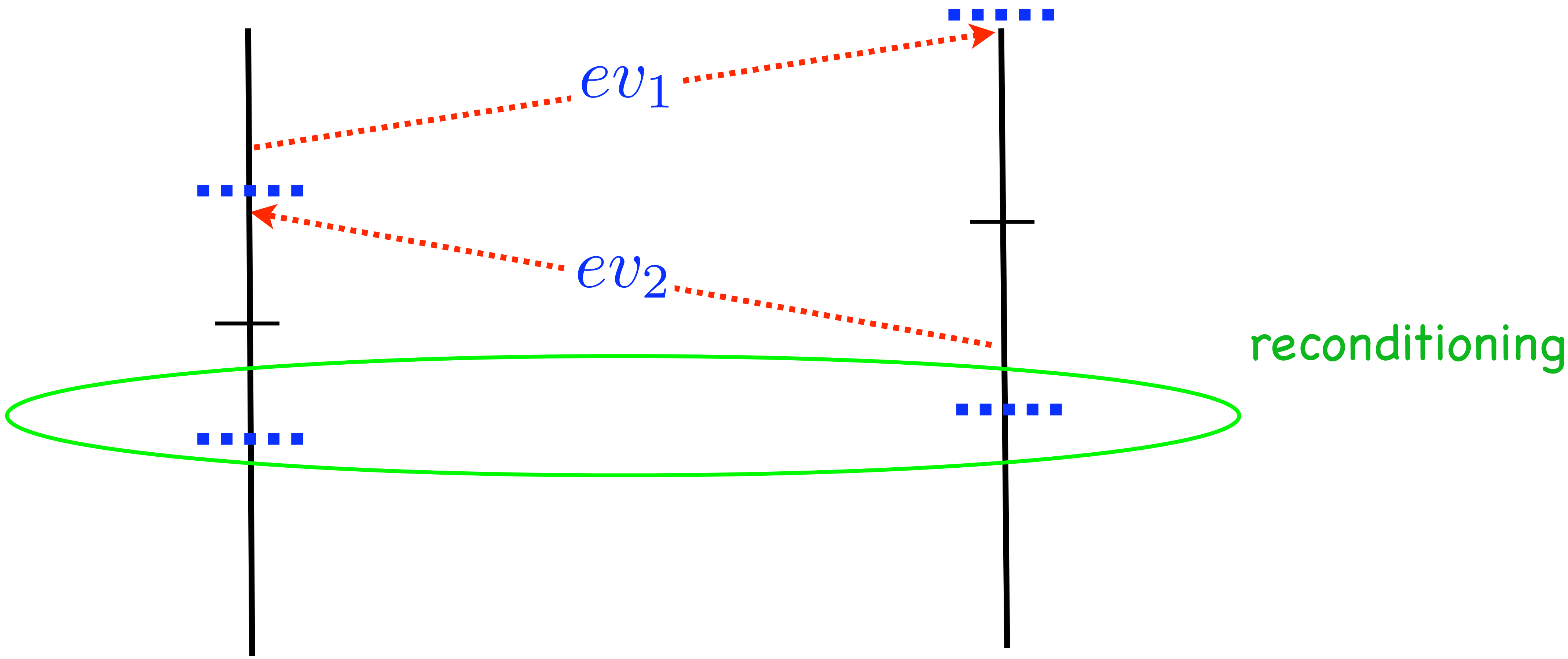
$$s_1 = \texttt{generate}\; ev_1; \; \texttt{await}\; ev_2; \; \texttt{cooperate}; \; \texttt{generate}\; ev_3 \quad \nmid \quad s_2 = \texttt{await}\; ev_1; \; \texttt{generate}\; ev_2; \; \texttt{await}\; ev_3$$



Synchronisation barrier

Reconditioning

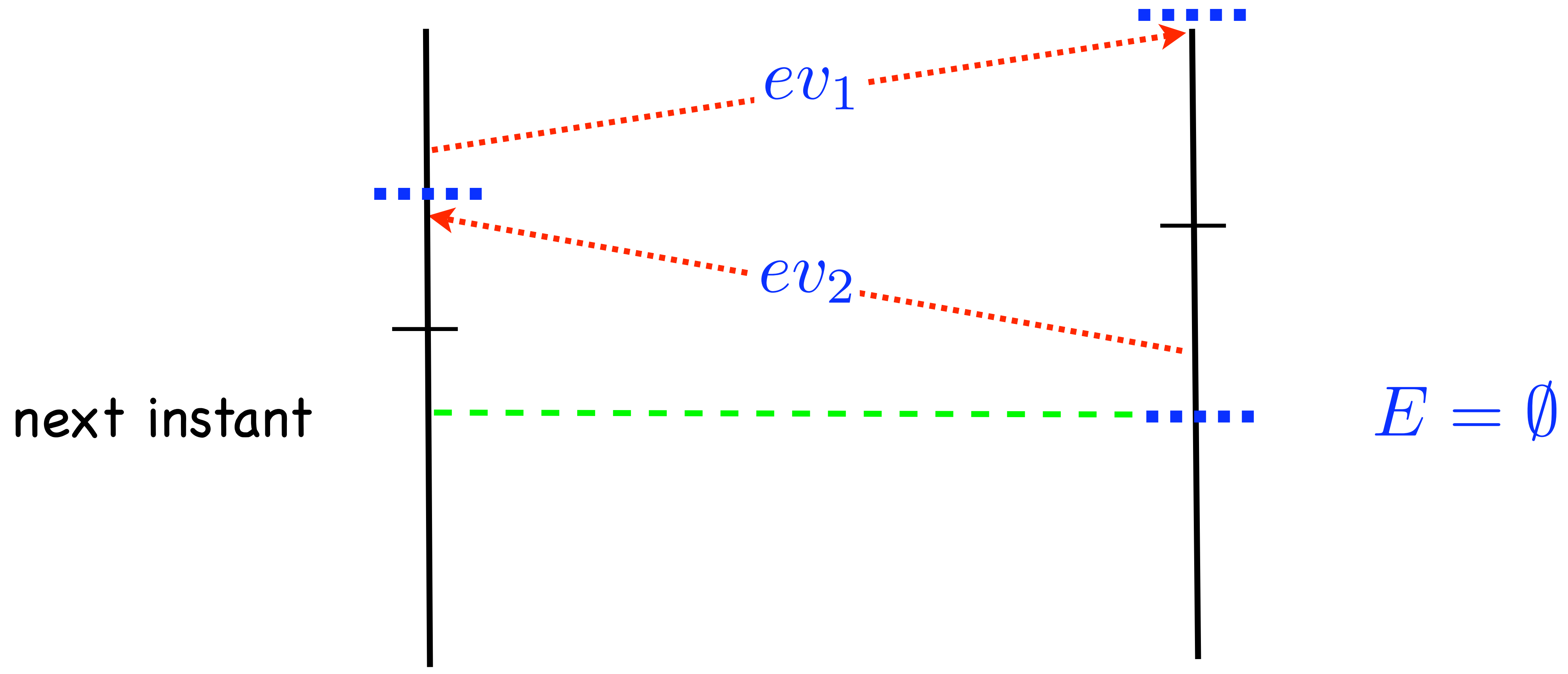
$$s_1 = \texttt{generate}\; ev_1; \;\texttt{await}\; ev_2; \;\texttt{cooperate}; \;\texttt{generate}\; ev_3 \quad \nmid \quad s_2 = \texttt{await}\; ev_1; \;\texttt{generate}\; ev_2; \;\texttt{await}\; ev_3$$



Synchronisation barrier

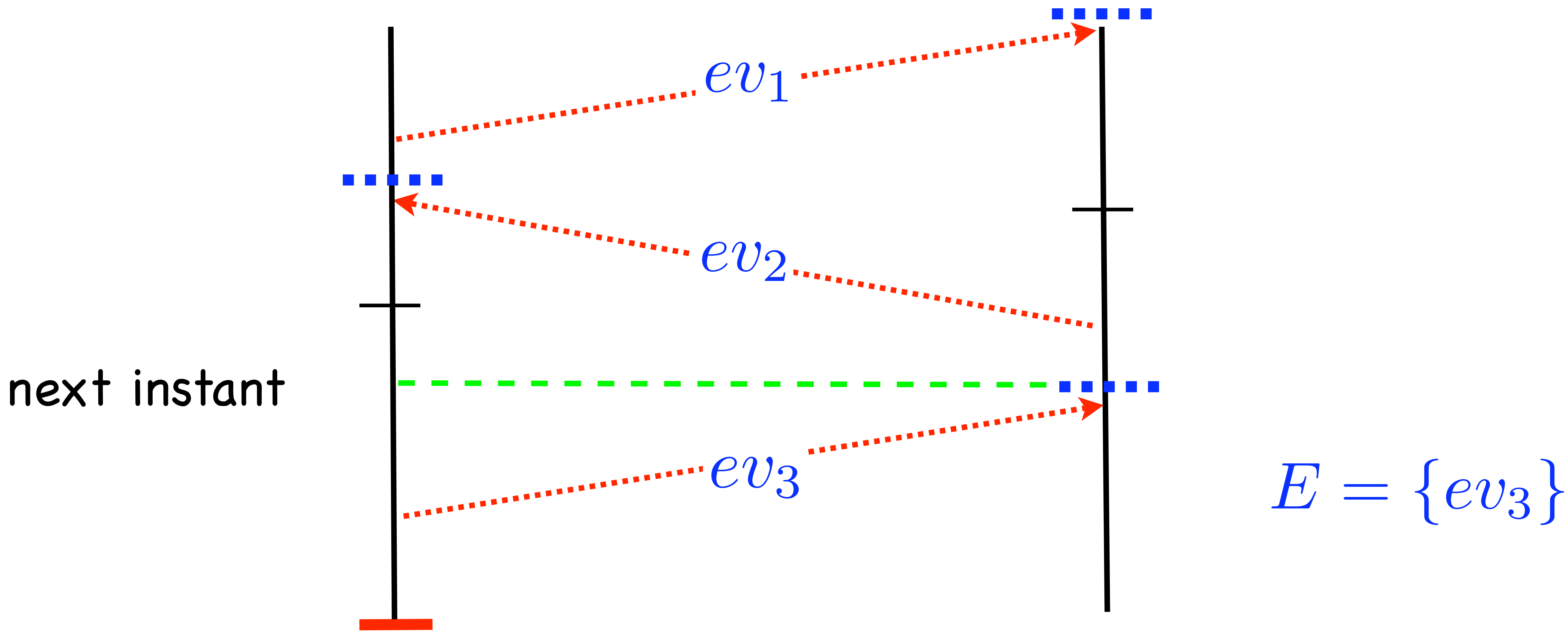
Instant passing

$$s_1 = \texttt{generate}\ ev_1;\ \texttt{await}\ ev_2;\ \texttt{generate}\ ev_3 \quad \nmid \quad s_2 = \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2;\ \texttt{await}\ ev_3$$



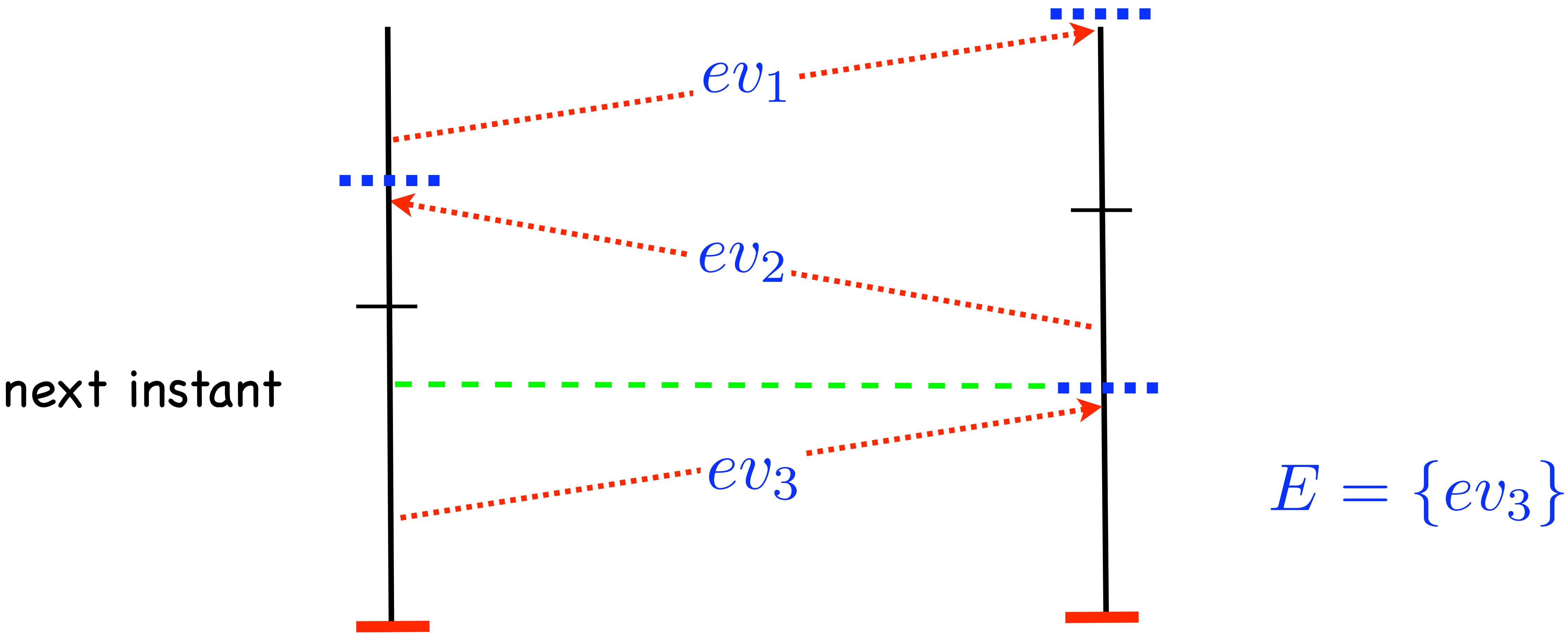
Next instant

$$s_1 = \texttt{generate}\ ev_1;\ \texttt{await}\ ev_2;\ \texttt{generate}\ ev_3 \quad \nmid \quad s_2 = \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2;\ \texttt{await}\ ev_3$$



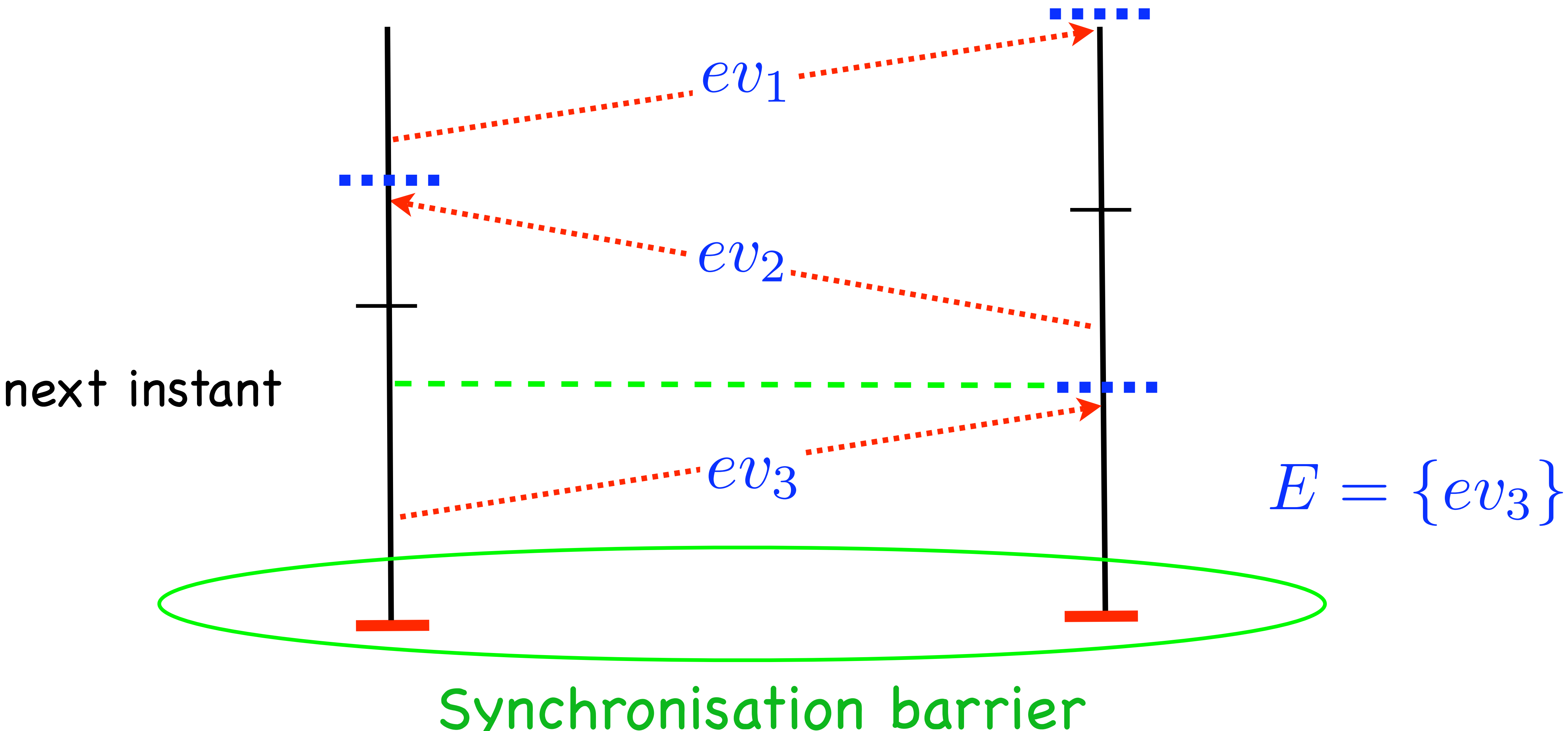
Next instant

$$s_1 = \texttt{generate}\ ev_1;\ \texttt{await}\ ev_2;\ \texttt{generate}\ ev_3 \quad \nmid \quad s_2 = \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2;\ \texttt{await}\ ev_3$$



Termination

$$s_1 \;=\; \texttt{generate}\ ev_1;\ \texttt{await}\ ev_2;\ \texttt{generate}\ ev_3 \quad \nmid \quad s_2 \;=\; \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2;\ \texttt{await}\ ev_3$$

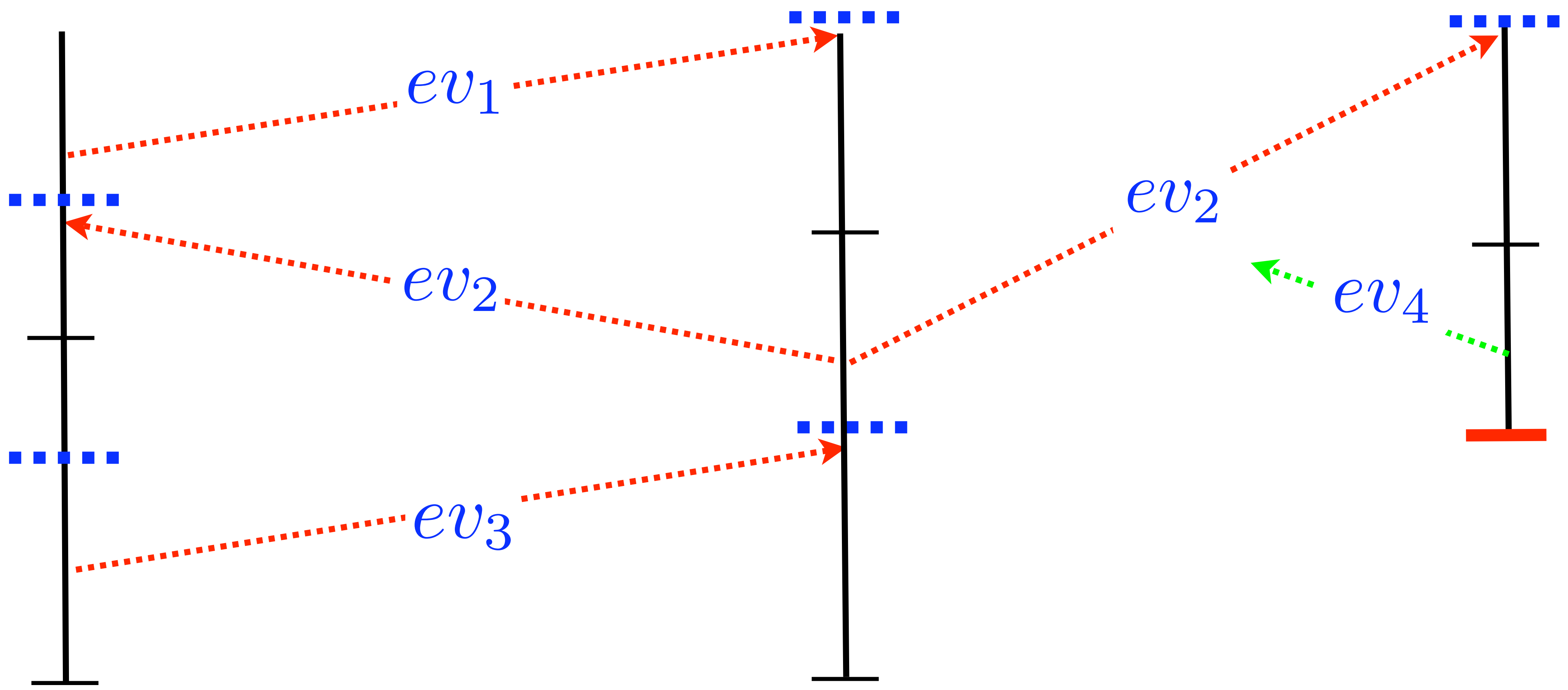


Time out

$$s'_1 \quad = \quad \texttt{do}\ s_1\ \texttt{watching}\ ev_4$$

$$s_2 \quad = \quad \texttt{await}\ ev_1;\ \texttt{generate}\ ev_2\ \texttt{await}\ ev_3$$

$$s_3 \quad = \quad \texttt{await}\ ev_2;\ \texttt{generate}\ ev_4$$

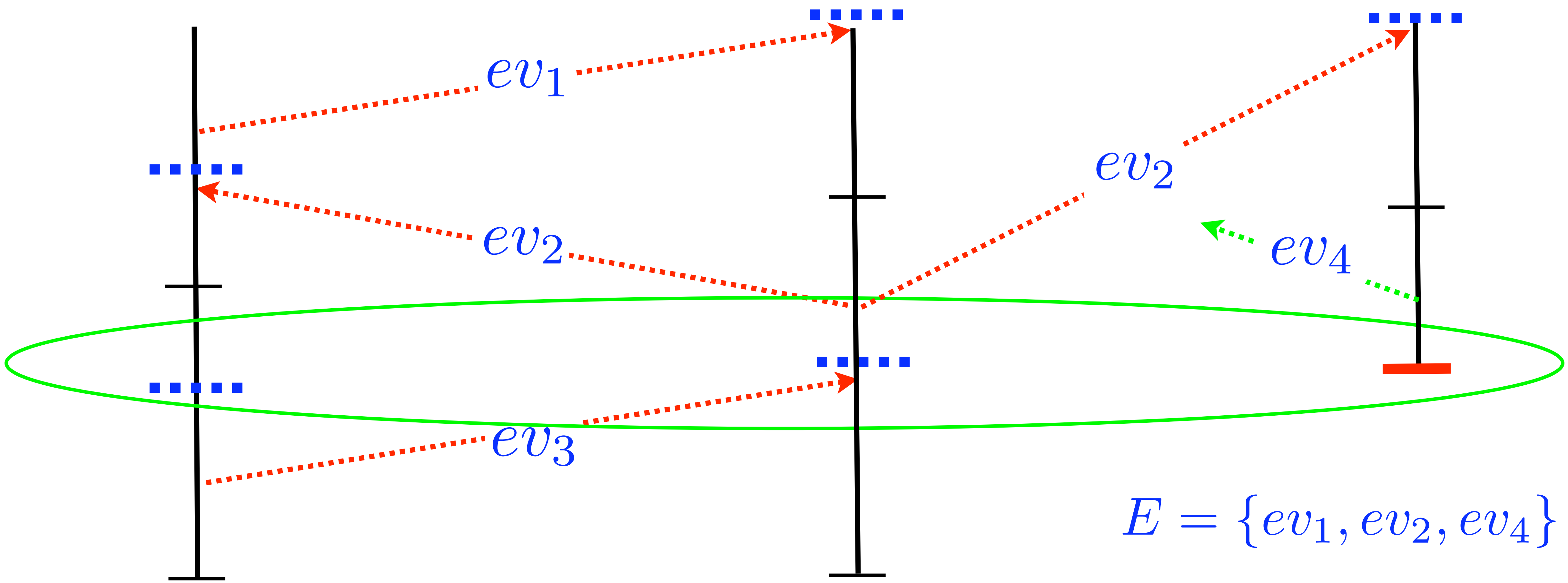


Time out

$$s_1'' = \texttt{do}\ (\texttt{cooperate;}\ \texttt{generate}\ ev_3)\ \texttt{watching}\ ev_4$$

$$s_2' = \texttt{await}\ ev_3$$

$$s_3' = \texttt{nothing}$$

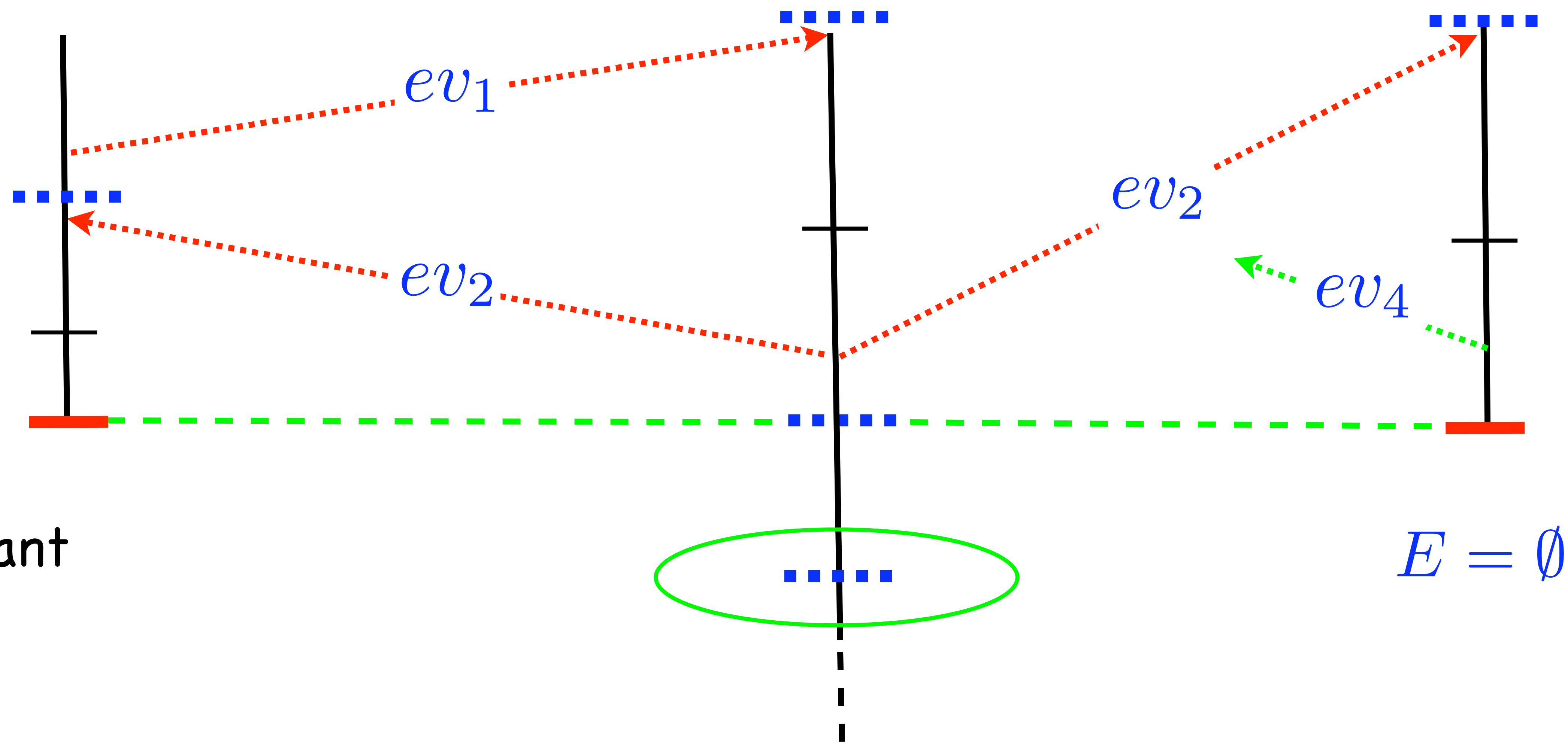


Reconditioning

$$s_1''' = \texttt{nothing}$$

$$s_2' = \texttt{await}\ ev_3$$

$$s_3' = \texttt{nothing}$$



next instant

$E = \emptyset$

Syntax of CRL

Expressions

$$exp ::= v \;\mid\; x \;\mid\; f(\overrightarrow{exp})$$

Programs

$$
\begin{aligned}
s ::= \quad & \texttt{nothing} \mid (\texttt{if}\ exp\ \texttt{then}\ s\ \texttt{else}\ s) \mid s; s \mid (s \nmid s) \mid \\
& \texttt{cooperate} \mid \texttt{generate}\ ev \mid \texttt{await}\ ev \mid \texttt{do}\ s\ \texttt{watching}\ ev \mid \\
& (\texttt{loop}\ s) \mid (\texttt{repeat}\ exp\ \texttt{do}\ s)
\end{aligned}
$$

Semantics of CRL

Event environment $E \subseteq Events$

Small-step transition relation:

$$\langle s, E \rangle \rightarrow \langle s', E' \rangle$$

Tick transition relation:

$$\langle s, E \rangle \hookrightarrow \langle [s]_E, \emptyset \rangle$$

Semantics: suspension

Suspension predicate $\langle s, E \rangle \ddagger \quad : \quad s$ is suspended in E.

$$\langle \mathit{cooperate}, E \rangle \ddagger \qquad (\mathit{coop})$$

$$\frac{ev \notin E}{\langle \texttt{await}\ ev, E \rangle \ddagger} \qquad (\mathit{wait}_s)$$

$$\frac{\langle s_1, E \rangle \ddagger}{\langle s_1 ; s_2, E \rangle \ddagger} \qquad (\mathit{seq}_s)$$

$$\frac{\langle s_1, E \rangle \ddagger \qquad \langle s_2, E \rangle \ddagger}{\langle s_1 \nmid s_2, E \rangle \ddagger} \qquad (\mathit{par}_s)$$

$$\frac{\langle s, E \rangle \ddagger}{\langle \texttt{do}\ s\ \texttt{watching}\ ev, E \rangle \ddagger} \qquad (\mathit{watch}_s)$$

Program reconditioning

Function $[s]_E$: erases guarding `cooperate`, kills "timed-out" `watching`.

$$[\texttt{cooperate}]_E = \texttt{nothing} \qquad [\texttt{await}\ ev]_E = \texttt{await}\ ev$$

$$[s_1 ; s_2]_E = [s_1]_E ; s_2 \qquad [s_1 \nmid s_2]_E = [s_1]_E \nmid [s_2]_E$$

$$[\texttt{do}\ s\ \texttt{watching}\ ev]_E = \begin{cases} \texttt{nothing} & if \quad ev \in E \\ \texttt{do}\ [s]_E\ \texttt{watching}\ ev & otherwise \end{cases}$$

Program reconditioning

Function $[s]_E$: erases guarding `cooperate`, kills "timed-out" `watching`.

$$[\texttt{cooperate}]_E = \texttt{nothing} \qquad [\texttt{await}\ ev]_E = \texttt{await}\ ev$$

$$[s_1 ; s_2]_E = [s_1]_E \,;\, s_2 \qquad [s_1 \nmid s_2]_E = [s_1]_E \nmid [s_2]_E$$

$$[\texttt{do}\ s\ \texttt{watching}\ ev]_E = \begin{cases} \texttt{nothing} & if \quad ev \in E \\ \texttt{do}\ [s]_E\ \texttt{watching}\ ev & otherwise \end{cases}$$

Tick transition relation:

$$\frac{\langle s, E \rangle \ddagger}{\langle s, E \rangle \hookrightarrow \langle [s]_E, \emptyset \rangle} \quad (tick)$$

Semantics: reactive operators

$$\langle \texttt{generate}\ ev, E\rangle \longrightarrow \langle \texttt{nothing}, E \cup \{ev\}\rangle \quad (gen)$$

$$\frac{ev \in E}{\langle \texttt{await}\ ev, E\rangle \longrightarrow \langle \texttt{nothing}, E\rangle} \quad (wait)$$

$$\frac{\langle s, E\rangle \longrightarrow \langle s', E'\rangle}{\langle \texttt{do}\ s\ \texttt{watching}\ ev, E\rangle \longrightarrow \langle \texttt{do}\ s'\ \texttt{watching}\ ev, E'\rangle} \quad (watch_1)$$

$$\langle \texttt{do nothing watching}\ ev, E\rangle \longrightarrow \langle \texttt{nothing}, E\rangle \quad (watch_2)$$

Semantics: sequence & parallel

$$\frac{\langle s_1, E\rangle \rightarrow \langle s_1', E'\rangle}{\langle s_1; s_2, E\rangle \rightarrow \langle s_1'; s_2, E'\rangle}\ (seq_1) \qquad \langle \texttt{nothing}\,; s, E\rangle \rightarrow \langle s, E\rangle\ \ (seq_2)$$

$$\frac{\langle s_1, E\rangle \rightarrow \langle s_1', E'\rangle}{\langle s_1 \nmid s_2, E\rangle \rightarrow \langle s_1' \nmid s_2, E'\rangle}\ (par_1) \qquad \langle \texttt{nothing} \nmid s, E\rangle \rightarrow \langle s, E\rangle\ \ (par_2)$$

$$\frac{\langle s_1, E\rangle \ddagger \quad \langle s_2, E\rangle \rightarrow \langle s_2', E'\rangle}{\langle s_1 \nmid s_2, E\rangle \rightarrow \langle s_1 \nmid s_2', E'\rangle}\ (par_3) \qquad \frac{\langle s, E\rangle \ddagger}{\langle s \nmid \texttt{nothing}, E\rangle \rightarrow \langle s, E\rangle}\ (par_4)$$

Semantics: loop/repeat

$$\langle \texttt{loop}\, s, E\rangle \rightarrow \langle (s \nmid \texttt{cooperate}); \texttt{loop}\, s, E\rangle \qquad (loop)$$

$$\frac{exp \rightsquigarrow n}{\langle \texttt{repeat}\, exp\, \texttt{do}\, s, E\rangle \rightarrow \langle \underbrace{s; \ldots; s}_{n\ times}, E\rangle} \qquad (repeat)$$

Semantics: conditional

$$\frac{exp \rightsquigarrow tt}{\langle \texttt{if}\ exp\ \texttt{then}\ s_1\ \texttt{else}\ s_2, E\rangle \longrightarrow \langle s_1, E\rangle} \quad (if_1)$$

$$\frac{exp \rightsquigarrow \mathit{ff}}{\langle \texttt{if}\ exp\ \texttt{then}\ s_1\ \texttt{else}\ s_2, E\rangle \longrightarrow \langle s_2, E\rangle} \quad (if_2)$$

Semantics: first properties

Determinism

$$s \neq \texttt{nothing} \quad \Rightarrow \quad \textit{either}\ \langle s, E\rangle \ddagger \quad \textit{or} \quad \exists!\, s', E' \,.\, \langle s, E\rangle \longrightarrow \langle s', E'\rangle$$

Event persistence

$$\langle s, E\rangle \longrightarrow \langle s', E'\rangle \quad \Rightarrow \quad E \subseteq E'$$

Semantics: first properties

Determinism

$$s \neq \texttt{nothing} \quad \Rightarrow \quad \textit{either} \; \langle s, E \rangle \ddagger \quad \textit{or} \;\; \exists!\, s', E' \,.\, \langle s, E \rangle \longrightarrow \langle s', E' \rangle$$

(because $\nmid$ is deterministic)

Event persistence

$$\langle s, E \rangle \longrightarrow \langle s', E' \rangle \quad \Rightarrow \quad E \subseteq E'$$

(because E is only changed by $\texttt{generate}\, ev$)

Convergence relations

Immediate convergence

$$\langle s, E \rangle \underset{\curlyvee}{\ddagger} \;\Leftrightarrow\; \langle s, E \rangle \ddagger \;\vee\; s = \texttt{nothing}$$

Instantaneous convergence ($\Rightarrow \;=_{\text{def}}\; \rightarrow^{*}$)

$$\langle s, E \rangle \Downarrow \langle s', E' \rangle \quad \text{if} \quad \langle s, E \rangle \Rightarrow \langle s', E' \rangle \;\wedge\; \langle s', E' \rangle \underset{\curlyvee}{\ddagger}$$

Instantaneous termination

$$\langle s, E \rangle \mathrel{\overline{\Downarrow}} E' \quad \text{if} \quad \langle s, E \rangle \Downarrow \langle \texttt{nothing}, E' \rangle$$

Convergence relations/predicates

Immediate convergence

$$\langle s, E \rangle \overset{\ddagger}{\curlyvee} \quad \Leftrightarrow \quad \langle s, E \rangle \ddagger \ \vee \ s = \texttt{nothing}$$

Instantaneous convergence

$$\langle s, E \rangle \Downarrow \langle s', E' \rangle \qquad \text{if} \quad \langle s, E \rangle \Rightarrow \langle s', E' \rangle \ \wedge \ \langle s', E' \rangle \overset{\ddagger}{\curlyvee}$$

$$\langle s, E \rangle \Downarrow \qquad \text{if} \quad \exists s', E' \, . \ \langle s, E \rangle \Downarrow \langle s', E' \rangle$$

Instantaneous termination

$$\langle s, E \rangle \Downarrow\!\!\!\!- \ E' \qquad \text{if} \quad \langle s, E \rangle \Downarrow \langle \texttt{nothing}, E' \rangle$$

$$\langle s, E \rangle \Downarrow\!\!\!\!- \qquad \text{if} \quad \exists\, E' \, . \ \langle s, E \rangle \Downarrow\!\!\!\!- \ E'$$

Semantics: more properties

Instantaneous size: $size(s)$

Size reduction within an instant

$$(\, \langle s, E \rangle \rightarrow \langle s', E' \rangle \;\Rightarrow\; size(s') < size(s) \,)$$

Reactivity (bounded by the size)

$$\forall s, \forall E \quad (\, \exists\, n \leq size(s) \,.\, \langle s, E \rangle \Downarrow_n \,)$$

instantaneous convergence in n steps

Semantics: more properties

Monotonicity

$$\langle s, E \rangle \Downarrow E' \Rightarrow \forall \hat{E} \supset E \;\; \exists \hat{E}' \supseteq E' \,.\;\; \langle s, \hat{E} \rangle \Downarrow \hat{E}'$$

Monotonicity of terminating computations

$$\langle s, E \rangle \Downarrow_n E' \Rightarrow \forall \hat{E} \supset E \;\; \exists \hat{E}' \supseteq E' \;\; \langle s, \hat{E} \rangle \Downarrow_n \hat{E}'$$

Bisimilarities

Two bisimulation equivalences of different granularity:

Fine-grained bisimulation: based on $\langle s, E\rangle \rightarrow \langle s', E'\rangle$

-> The observer is a thread

Coarse-grained bisimulation: based on $\langle s, E\rangle \Downarrow \langle s', E'\rangle$

-> The observer is the environment

Fine-grained bisimilarity

A symmetric $\mathcal{R}$ is a *fg-bisimulation* if $s_1 \, \mathcal{R} \, s_2$ implies, for any $E \in Events$:

$$1) \ \langle s_1, E \rangle \rightarrow \langle s_1', E' \rangle \quad \Rightarrow \quad \langle s_2, E \rangle \Rightarrow \langle s_2', E' \rangle \ \wedge \ s_1' \, \mathcal{R} \, s_2'$$

$$2) \ \langle s_1, E \rangle \ddagger \quad \Rightarrow \quad \langle s_2, E \rangle \Downarrow \langle s_2', E \rangle \ \wedge \ \llcorner s_1 \lrcorner_E \, \mathcal{R} \, \llcorner s_2' \lrcorner_E$$

fg-bisimilarity : $s_1 \approx^{fg} s_2$ if $s_1 \, \mathcal{R} \, s_2$ for some fg-bisimulation $\mathcal{R}$.

Notation $\quad \llcorner s \lrcorner_E \stackrel{\text{def}}{=} \begin{cases} [s]_E & \text{if } \langle s, E \rangle \ddagger \\ s & \text{otherwise} \end{cases}$

reconditioning extended to non-suspended programs

Fine-grained bisimilarity

A symmetric $\mathcal{R}$ is a *fg-bisimulation* if $s_1 \, \mathcal{R} \, s_2$ implies, for any $E \in \mathit{Events}$:

1) $\langle s_1, E\rangle \rightarrow \langle s_1', E'\rangle \quad \Rightarrow \quad \langle s_2, E\rangle \Rightarrow \langle s_2', E'\rangle \;\wedge\; s_1' \, \mathcal{R} \, s_2'$

2) $\langle s_1, E\rangle \ddagger \quad \Rightarrow \quad \langle s_2, E\rangle \Downarrow \langle s_2', E\rangle \;\wedge\; \llcorner s_1 \lrcorner_E \, \mathcal{R} \, \llcorner s_2' \lrcorner_E$

fg-bisimilarity : $s_1 \approx^{fg} s_2$ if $s_1 \, \mathcal{R} \, s_2$ for some fg-bisimulation $\mathcal{R}$.

Fine-grained bisimilarity is time-insensitive (weak) and termination insensitive:

$$(\texttt{nothing}\,;\,\texttt{generate}\,ev) \quad \approx^{fg} \quad \texttt{generate}\,ev$$

$$\texttt{nothing} \quad \approx^{fg} \quad \texttt{cooperate} \quad \approx^{fg} \quad \texttt{loop}\,\texttt{nothing}$$

Fine-grained bisimilarity

A symmetric $\mathcal{R}$ is a *fg-bisimulation* if $s_1 \, \mathcal{R} \, s_2$ implies, for any $E \in Events$:

$$1) \ \langle s_1, E\rangle \rightarrow \langle s_1', E'\rangle \quad \Rightarrow \quad \langle s_2, E\rangle \Rightarrow \langle s_2', E'\rangle \ \wedge \ s_1' \, \mathcal{R} \, s_2'$$

$$2) \ \langle s_1, E\rangle \ddagger \quad \Rightarrow \quad \langle s_2, E\rangle \Downarrow \langle s_2', E\rangle \ \wedge \ \llcorner s_1 \lrcorner_E \, \mathcal{R} \, \llcorner s_2' \lrcorner_E$$

fg-bisimilarity : $s_1 \approx^{fg} s_2$ if $s_1 \, \mathcal{R} \, s_2$ for some fg-bisimulation $\mathcal{R}$.

Fine-grained bisimilarity does not preserve tick transitions:

$$\texttt{nothing} \ \approx^{fg} \ \texttt{cooperate} \ \approx^{fg} \ \texttt{loop}\,\texttt{nothing}$$

but it preserves the clock-stamp of events:

$$\texttt{nothing}\,;\,\texttt{generate}\,ev \ \not\approx^{fg} \ \texttt{cooperate}\,;\,\texttt{generate}\,ev$$

Coarse-grained bisimilarity

A symmetric $\mathcal{R}$ is a *cg-bisimulation* if $s_1 \,\mathcal{R}\, s_2$ implies, for any $E \in \mathit{Events}$:

$$\langle s_1, E\rangle \Downarrow \langle s_1', E'\rangle \;\Rightarrow\; \langle s_2, E\rangle \Downarrow \langle s_2', E'\rangle \;\wedge\; \llcorner s_1' \lrcorner_{E'} \;\mathcal{R}\; \llcorner s_2' \lrcorner_{E'}$$

cg-bisimilarity : $s_1 \approx^{cg} s_2$ if $s_1 \,\mathcal{R}\, s_2$ for some cg-bisimulation $\mathcal{R}$.

Coarse-grained bisimilarity preserves the overall effect of instant computations

Coarse-grained bisimilarity

A symmetric $\mathcal{R}$ is a *cg-bisimulation* if $s_1 \, \mathcal{R} \, s_2$ implies, for any $E \in Events$:

$$\langle s_1, E \rangle \Downarrow \langle s_1', E' \rangle \;\; \Rightarrow \;\; \langle s_2, E \rangle \Downarrow \langle s_2', E' \rangle \; \wedge \; \llcorner s_1' \lrcorner_{E'} \; \mathcal{R} \; \llcorner s_2' \lrcorner_{E'}$$

cg-bisimilarity : $s_1 \approx^{cg} s_2$ if $s_1 \, \mathcal{R} \, s_2$ for some cg-bisimulation $\mathcal{R}$.

Coarse-grained bisimilarity preserves the overall effect of instant computations

-> makes sense in combination with reactivity

Coarse-grained bisimilarity

A symmetric $\mathcal{R}$ is a *cg-bisimulation* if $s_1 \, \mathcal{R} \, s_2$ implies, for any $E \in Events$:

$$\langle s_1, E\rangle \Downarrow \langle s'_1, E'\rangle \quad \Rightarrow \quad \langle s_2, E\rangle \Downarrow \langle s'_2, E'\rangle \;\wedge\; \llcorner s'_1 \lrcorner_{E'} \; \mathcal{R} \; \llcorner s'_2 \lrcorner_{E'}$$

cg-bisimilarity : $s_1 \approx^{cg} s_2$ if $s_1 \, \mathcal{R} \, s_2$ for some cg-bisimulation $\mathcal{R}$.

Coarse-grained bisimilarity is strictly larger than fine-grained bisimilarity:

$$\approx^{fg} \;\subset\; \approx^{cg}$$

Examples

$\approx^{cg}$ is more abstract than $\approx^{fg}$ because:

$\approx^{cg}$ is generation-order-insensitive:

$$(\texttt{generate}\; ev_1 \nmid \texttt{generate}\; ev_2) \quad \approx^{cg} \quad (\texttt{generate}\; ev_2 \nmid \texttt{generate}\; ev_1)$$

$$(\texttt{generate}\; ev_1 \nmid \texttt{generate}\; ev_2) \quad \not\approx^{fg} \quad (\texttt{generate}\; ev_2 \nmid \texttt{generate}\; ev_1)$$

$\approx^{cg}$ is stuttering-insensitive:

$$\texttt{generate}\; ev \;\approx^{cg}\; (\texttt{generate}\; ev \,;\, \texttt{generate}\; ev)$$

$$\texttt{generate}\; ev \;\not\approx^{fg}\; (\texttt{generate}\; ev \,;\, \texttt{generate}\; ev)$$

Properties

Commutativity of $\nmid$ with respect to $\approx^{cg}$:

$$s_1 \nmid s_2 \approx^{cg} s_2 \nmid s_1$$

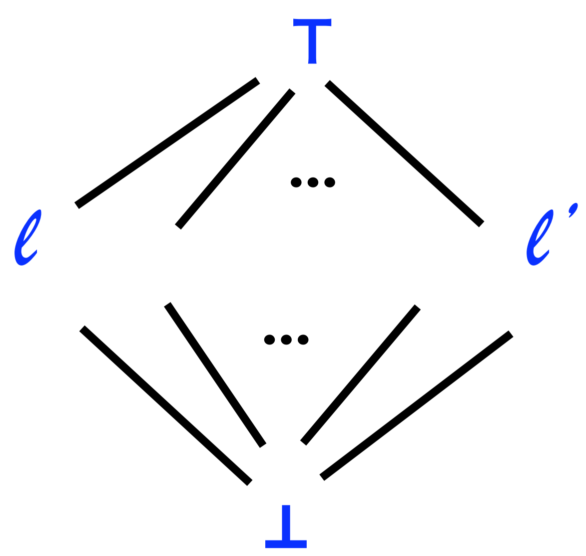
Compositionality of both $\approx^{cg}$ and $\approx^{fg}$ with respect to $\nmid$

Secure information flow

Goal: information flow control in CRL enriched with security levels for variables and events

A finite lattice of security levels :

levels assigned to variables and events



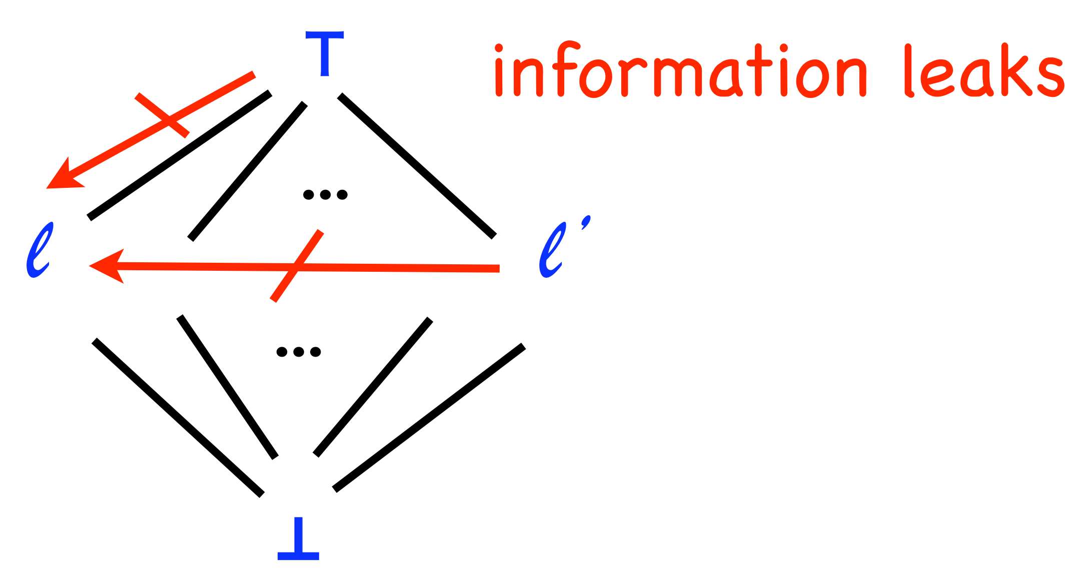
Secure information flow : generated events of level ℓ only depend on tested variables or tested events of level ℓ_0 with $\ell_0 \leq \ell$

Secure information flow

Goal: information flow control in CRL enriched with security levels for variables and events

A finite lattice of security levels :

levels assigned to variables and events



Secure information flow : generated events of level ℓ only depend on tested variables or tested events of level ℓ_0 with $\ell_0 \leq \ell$

$\Gamma\mathcal{L}$-observation

Lattice of security levels : $(\mathcal{S}, \leq)$ $\mathcal{L} \subseteq \mathcal{S}$ downward-closed

$\mathcal{L}$-observer : sees only objects of level in $\mathcal{L}$

Type environment : $\Gamma : Var \cup Events \rightarrow \mathcal{S}$

Valuation : $V : Var \rightarrow Val$

$\Gamma\mathcal{L}$-equality of valuations and event environments

$$V_1 =^{\Gamma}_{\mathcal{L}} V_2 \quad \textit{if} \quad \Gamma(x) \in \mathcal{L} \Rightarrow V_1(x) = V_2(x)$$

$$E_1 =^{\Gamma}_{\mathcal{L}} E_2 \quad \textit{if} \quad \Gamma(ev) \in \mathcal{L} \Rightarrow (\, ev \in E_1 \Leftrightarrow ev \in E_2 \,)$$

Fine-grained $\Gamma\mathcal{L}$-bisimilarity

$\mathcal{R}$ is a fg-$\Gamma\mathcal{L}$-V_1V_2-bisimulation if $s_1 \,\mathcal{R}\, s_2$ implies, for any $E_1 \;=^{\Gamma}_{\mathcal{L}}\; E_2$:

1) $\langle s_1, E_1\rangle \rightarrow_{V_1} \langle s_1', E_1'\rangle \;\Rightarrow\; (\,\langle s_2, E_2\rangle \Rightarrow_{V_2} \langle s_2', E_2'\rangle \;\wedge\; E_1' =^{\Gamma}_{\mathcal{L}} E_2' \;\wedge\; s_1' \,\mathcal{R}\; s_2'\,)$

2) $\langle s_1, E_1\rangle \ddagger \;\Rightarrow\; (\,\langle s_2, E_2\rangle \Downarrow_{V_2} \langle s_2', E_2'\rangle \;\wedge\; E_1 =^{\Gamma}_{\mathcal{L}} E_2' \;\wedge\; \llcorner s_1 \lrcorner_{E_1} \;\mathcal{R}\; \llcorner s_2' \lrcorner_{E_2'}\,)$

3) and 4) : Symmetric clauses for $\langle s_2, E_2\rangle$ under valuation V_2.

fg-$\Gamma\mathcal{L}$-bisimilarity: $s_1 \approx^{fg}_{\Gamma\mathcal{L}} s_2$ if for any $V_1 \;=^{\Gamma}_{\mathcal{L}}\; V_2$, $s_1 \,\mathcal{R}\, s_2$ for some…

Fine-grained $\Gamma\mathcal{L}$-bisimilarity

$\mathcal{R}$ is a fg-$\Gamma\mathcal{L}$-V_1V_2-bisimulation if $s_1 \,\mathcal{R}\, s_2$ implies, for any $E_1 \; =^{\Gamma}_{\mathcal{L}} \; E_2$:

1) $\langle s_1, E_1\rangle \rightarrow_{V_1} \langle s_1', E_1'\rangle \;\Rightarrow\; (\,\langle s_2, E_2\rangle \Rightarrow_{V_2} \langle s_2', E_2'\rangle \;\wedge\; E_1' =^{\Gamma}_{\mathcal{L}} E_2' \;\wedge\; s_1' \,\mathcal{R}\; s_2'\,)$

2) $\langle s_1, E_1\rangle \ddagger \;\;\Rightarrow\;\; (\,\langle s_2, E_2\rangle \Downarrow_{V_2} \langle s_2', E_2'\rangle \;\wedge\; E_1 =^{\Gamma}_{\mathcal{L}} E_2' \;\wedge\; \llcorner s_1 \lrcorner_{E_1} \;\mathcal{R}\; \llcorner s_2' \lrcorner_{E_2'}\,)$

3) and 4) : Symmetric clauses for $\langle s_2, E_2\rangle$ under valuation V_2.

fg-$\Gamma\mathcal{L}$-bisimilarity: $s_1 \approx^{fg}_{\Gamma\mathcal{L}} s_2$ if for any $V_1 \; =^{\Gamma}_{\mathcal{L}} \; V_2$, $s_1 \,\mathcal{R}\, s_2$ for some...

Fine-grained RNI

s is fg-secure in Γ if $s \approx^{fg}_{\Gamma\mathcal{L}} s$ for every $\mathcal{L}$.

Coarse-grained $\Gamma\mathcal{L}$-bisimilarity

$\mathcal{R}$ is a cg-$\Gamma\mathcal{L}$-$V_1 V_2$-bisimulation if $s_1 \,\mathcal{R}\, s_2$ implies, for any $E_1 =^{\Gamma}_{\mathcal{L}} E_2$:

1) $\langle s_1, E_1\rangle \Downarrow_{V_1} \langle s_1', E_1'\rangle \;\Rightarrow\; (\,\langle s_2, E_2\rangle \Downarrow_{V_2} \langle s_2', E_2'\rangle \wedge E_1' =^{\Gamma}_{\mathcal{L}} E_2' \wedge$
$\llcorner s_1' \lrcorner_{E_1'} \;\mathcal{R}\; \llcorner s_2' \lrcorner_{E_2'}\,)$

2) Symmetric clauses for $\langle s_2, E_2\rangle$ under valuation V_2.

cg-$\Gamma\mathcal{L}$-bisimilarity: $s_1 \approx^{cg}_{\Gamma\mathcal{L}} s_2$ if for any $V_1 =^{\Gamma}_{\mathcal{L}} V_2$, $s_1 \,\mathcal{R}\, s_2$ for some…

Coarse-grained $\Gamma\mathcal{L}$-bisimilarity

$\mathcal{R}$ is a cg-$\Gamma\mathcal{L}$-V_1V_2-bisimulation if $s_1\,\mathcal{R}\,s_2$ implies, for any $E_1 =^{\Gamma}_{\mathcal{L}} E_2$:

1) $\langle s_1, E_1\rangle \Downarrow_{V_1} \langle s_1', E_1'\rangle \;\Rightarrow\; (\,\langle s_2, E_2\rangle \Downarrow_{V_2} \langle s_2', E_2'\rangle \wedge E_1' =^{\Gamma}_{\mathcal{L}} E_2' \wedge \llcorner s_1' \lrcorner_{E_1'}\;\mathcal{R}\;\llcorner s_2' \lrcorner_{E_2'}\,)$

2) Symmetric clauses for $\langle s_2, E_2\rangle$ under valuation V_2.

cg-$\Gamma\mathcal{L}$-bisimilarity: $s_1 \approx^{cg}_{\Gamma\mathcal{L}} s_2$ if for any $V_1 =^{\Gamma}_{\mathcal{L}} V_2$, $s_1\,\mathcal{R}\,s_2$ for some...

Coarse-grained RNI

s is cg-secure in Γ if $s \approx^{cg}_{\Gamma\mathcal{L}} s$ for every $\mathcal{L}$.

Relation between the RNI's

If s is *fg*-secure then s is *cg*-secure.

cg-secure but not *fg*-secure:

$$s = \texttt{if}\ x^{\top} = 0 \quad \texttt{then}\ \texttt{generate}\ ev_1^{\perp} \nmid \texttt{generate}\ ev_2^{\perp}$$
$$\texttt{else}\ \texttt{generate}\ ev_2^{\perp} \nmid \texttt{generate}\ ev_1^{\perp}$$

Relation between the RNI's

If s is *fg*-secure then s is *cg*-secure.

cg-secure but not *fg*-secure:

$$s = \texttt{if}\ x^{\top} = 0 \quad \texttt{then}\ \texttt{generate}\ ev_1^{\perp} \nmid \texttt{generate}\ ev_2^{\perp}$$
$$\texttt{else}\ \texttt{generate}\ ev_2^{\perp} \nmid \texttt{generate}\ ev_1^{\perp}$$

Both *fg*-secure and *cg*-secure:

$$\texttt{if}\ x^{\top} = 0 \quad \texttt{then}\ \texttt{generate}\ ev_1^{\perp} \nmid \texttt{generate}\ ev_2^{\perp}$$
$$\texttt{else}\ \texttt{generate}\ ev_1^{\perp}\ ;\ \texttt{generate}\ ev_2^{\perp}\ \texttt{end}$$

Security type system

Prevent level drop from a tested variable/event to a generated event

$$\texttt{if } x^{\top} = 0 \quad \texttt{then nothing}$$
$$\texttt{else generate } ev_{\perp}$$

$$\texttt{do} \,(\, \texttt{cooperate} \,;\, \texttt{generate } ev_2^{\perp} \,)\, \texttt{watching } ev_1^{\top}$$

NB: in these cases the level drop happens within the same command.

Termination leaks

Leaks due to different termination behaviours depending on a high test

Prevent level drop after a high fork towards ≠ termination behaviours

Ex: low output after high conditional with a finite and an infinite branch

$$\texttt{if } x^{\top} = 0 \quad \texttt{then nothing}$$
$$\texttt{else loop nothing };$$
$$\texttt{generate } ev_{\perp}$$

Termination leaks

Prevent level drop after high conditionals

$\texttt{if } x^\top = 0 \quad \texttt{then nothing}$

$\qquad\qquad\quad \texttt{else loop nothing}\ ;$

$\texttt{generate } ev_\bot$

... and more generally after high tests leading to ≠ termination behaviours

$\texttt{await } ev_1^\top\ ;\ \texttt{generate } ev_2^\bot$

NB: in these cases the level drop spans over two subsequent commands.

Termination leaks

Prevent level drop after high conditionals

$$\texttt{if } x^{\top} = 0 \quad \texttt{then nothing}$$
$$\texttt{else loop nothing ;}$$
$$\texttt{generate } ev_{\perp}$$

First solution [Volpano & Smith 1998]: reject high conditionals.

Later solution: forbid loops in branches of high conditionals.

Too drastic! What matters is the absence of low outputs afterwards.

Termination leaks

Prevent level drop after high conditionals

$$\texttt{if}\ x^{\top} = 0 \quad \texttt{then nothing}$$
$$\texttt{else loop nothing}\ ;$$
$$\texttt{generate}\ ev_{\perp}$$

More permissive solution [Boudol and C., Smith 2001]: use double types

$$\Gamma \vdash s : (\tau, \sigma)$$

lower bound for writes — upper bound for reads

Termination leaks

Prevent level drop after high conditionals

$$\texttt{if } x^{\top} = 0 \quad \texttt{then nothing} \quad \texttt{else loop nothing} \;;$$

$$\texttt{generate } ev_{\bot}$$

$$(\textsc{Cond1}) \quad \frac{\Gamma \vdash exp : \vartheta, \quad \Gamma \vdash s_i : (\tau, \sigma), \qquad i = 1, 2, \quad \vartheta \leq \tau}{\Gamma \vdash \texttt{if } exp \texttt{ then } s_1 \texttt{ else } s_2 : (\tau, \vartheta \sqcup \sigma)}$$

$$(\textsc{Seq}) \quad \frac{\Gamma \vdash s_1 : (\tau_1, \sigma_1), \quad \Gamma \vdash s_2 : (\tau_2, \sigma_2), \quad \sigma_1 \leq \tau_2}{\Gamma \vdash s_1 \,;\, s_2 : (\tau_1 \sqcap \tau_2, \sigma_1 \sqcup \sigma_2)}$$

Termination leaks

Prevent level drop after high conditionals : $\Gamma \vdash s : (\tau, \sigma)$

However this solution still rules out secure programs:

$\texttt{if}\ x^{\top} = 0 \quad \texttt{then generate}\ ev_1^{\top}$
$\quad\quad\quad\quad\quad\quad \texttt{else nothing}\,;$
$\texttt{generate}\ ev_2^{\perp}$

Further refined solution:

$FIN = terminating$ programs, built without $\texttt{await}\ ev$, $\texttt{cooperate}$ and $\texttt{loop}$.

$INF = nonterminating$ programs, always entering a loop.

Termination leaks

Prevent level drop after non uniform high conditionals

$$(\textsc{Cond1})\quad \frac{\Gamma \vdash exp : \vartheta, \quad \Gamma \vdash s_i : (\tau, \sigma), \qquad i = 1,2, \quad \vartheta \leq \tau}{\Gamma \vdash \texttt{if}\ exp\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : (\tau, \vartheta \sqcup \sigma)}$$

$$(\textsc{Cond2})\quad \frac{\Gamma \vdash exp : \vartheta, \quad (\Gamma \vdash s_i : (\tau, \sigma) \quad \wedge \quad s_i \in FIN, \quad i = 1,2\,), \quad \vartheta \leq \tau}{\Gamma \vdash \texttt{if}\ exp\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : (\tau, \sigma)}$$

$$(\textsc{Cond3})\quad \frac{\Gamma \vdash exp : \vartheta, \quad (\Gamma \vdash s_i : (\tau, \sigma) \quad \wedge \quad s_i \in INF, \quad i = 1,2\,), \quad \vartheta \leq \tau}{\Gamma \vdash \texttt{if}\ exp\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : (\tau, \sigma)}$$

Security type system

Soundness of type system for fg-security

If s is typable in Γ then s is *fg*-secure in Γ.

Security type system

Soundness of type system for fg-security

If s is typable in Γ then s is *fg*-secure in Γ.

Soundness of type system for cg-security

If s is typable in Γ then s is *cg*-secure in Γ.

Related work

- Builds on [Almeida Matos, Boudol and Castellani, 2007] and previous work on synchronous languages [Boussinot, Susini, Amadio, Dabrowski, ...]. Main improvements with respect to [AMBC07]: ``better" left-parallel operator (associative, no scheduling leaks), reactivity, bisimilarities, coarse-grained security notion, more refined type system with precise treatment of termination leaks.
- [Sabelfeld, Russo 2007]: cooperative scheduling.
- [Bohannon et al. 2009]: ID-security.

References

2. A. Almeida Matos, G. Boudol, and I. Castellani. Typing noninterference for reactive programs. *Journal of Logic and Algebraic Programming*, 72(2):124–156, 2007.
3. R. M Amadio and F. Dabrowski. Feasible reactivity for synchronous cooperative threads. *Electronic Notes in Theoretical Computer Science*, 154(3):33–43, 2006.
4. Roberto M Amadio. The SL synchronous language, revisited. *The Journal of Logic and Algebraic Programming*, 70(2):121–150, 2007.
5. P. Attar and I. Castellani. Fine-grained and coarse-grained reactive noninterference, 2013. Forthcoming Research Report.
6. G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Sci. of Comput. Programming*, 19, 1992.
7. A. Bohannon, B. C. Pierce, V. Sjöberg, S. Weirich, and S. Zdancewic. Reactive noninterference. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 79–90. ACM, 2009.

Future work

- Extension to fully-fledged distributed reactive language, with memory, sites and migration (GALS) => mix of distribution, synchrony and asynchrony.

- More "practical" notions of security, allowing for declassification, e.g. using time-out (watching) to trigger declassification.

- Determinism: alternative trace-based definitions for behavioural equivalence and security.

Open questions

- Generality. How general is such a model of concurrent computation proceeding in successive phases, where programs interact in a constrained/disciplined way, recovering advantages of sequential computation? Analogy with membrane systems and session calculi.

- Expressiveness. Identify witness problems, naturally expressible in one model/language but not in others. [Witness in our model: hot-plug service replacement]. Then define a natural encoding from L1 to L2 as one that preserves both witness problems and some representative native operators (like ||).

Thank you!