Type-safe Eventful Sessions in Java

Raymond Hu, Dimitrios Kouzapas,
Olivier Pernet, Nobuko Yoshida

Imperial College London

Kohei Honda

Queen Mary University of London

Type-safe Eventful Sessions in Java

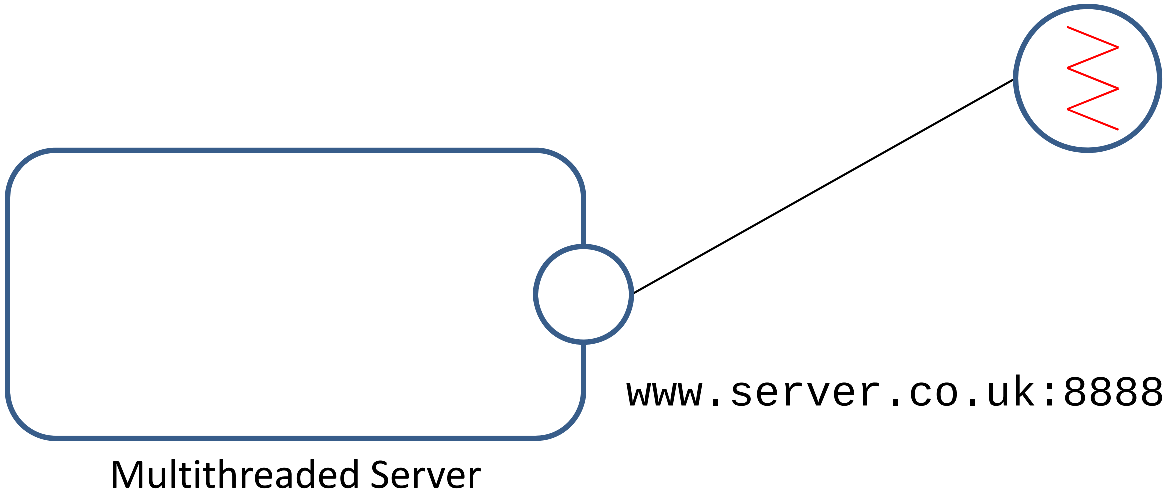
- Combine ***session types*** and ***event-driven programming***
  - Extend session types to support event-driven programming
  - Facilitate event-driven programming using the benefits of session types

Contributions

- Extend SJ (Session Java) for *session-typed event programming*
- Formalise the key mechanisms in a minimal process model; encoding of high-level event constructs; prove *communication safety* and *event progress* for event-driven session processes
- Implementation of run-time session type monitoring and transport-independent selector services
- Real-world applications: SMTP server; benchmarks

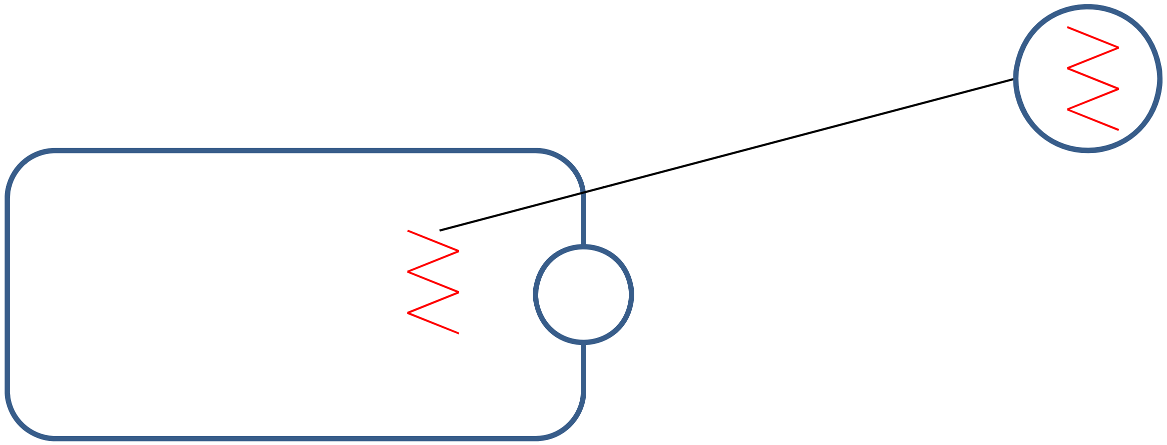
Event-driven Concurrency

- … vs. multithreaded concurrency



Event-driven Concurrency

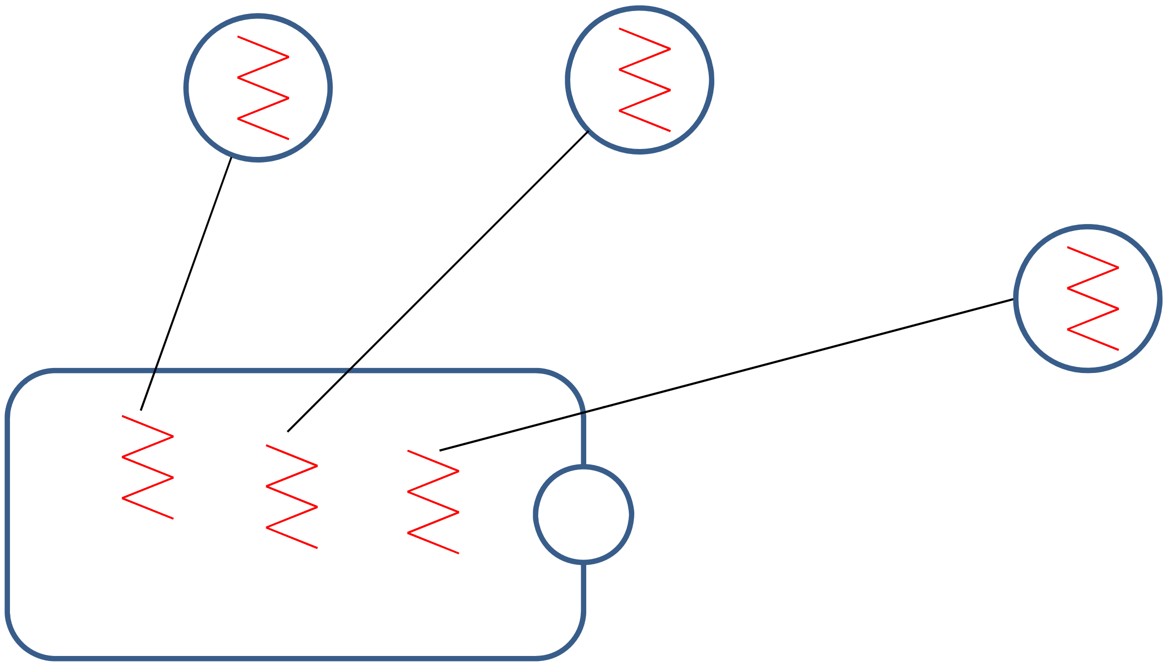
- … vs. multithreaded concurrency



Multithreaded Server

Event-driven Concurrency

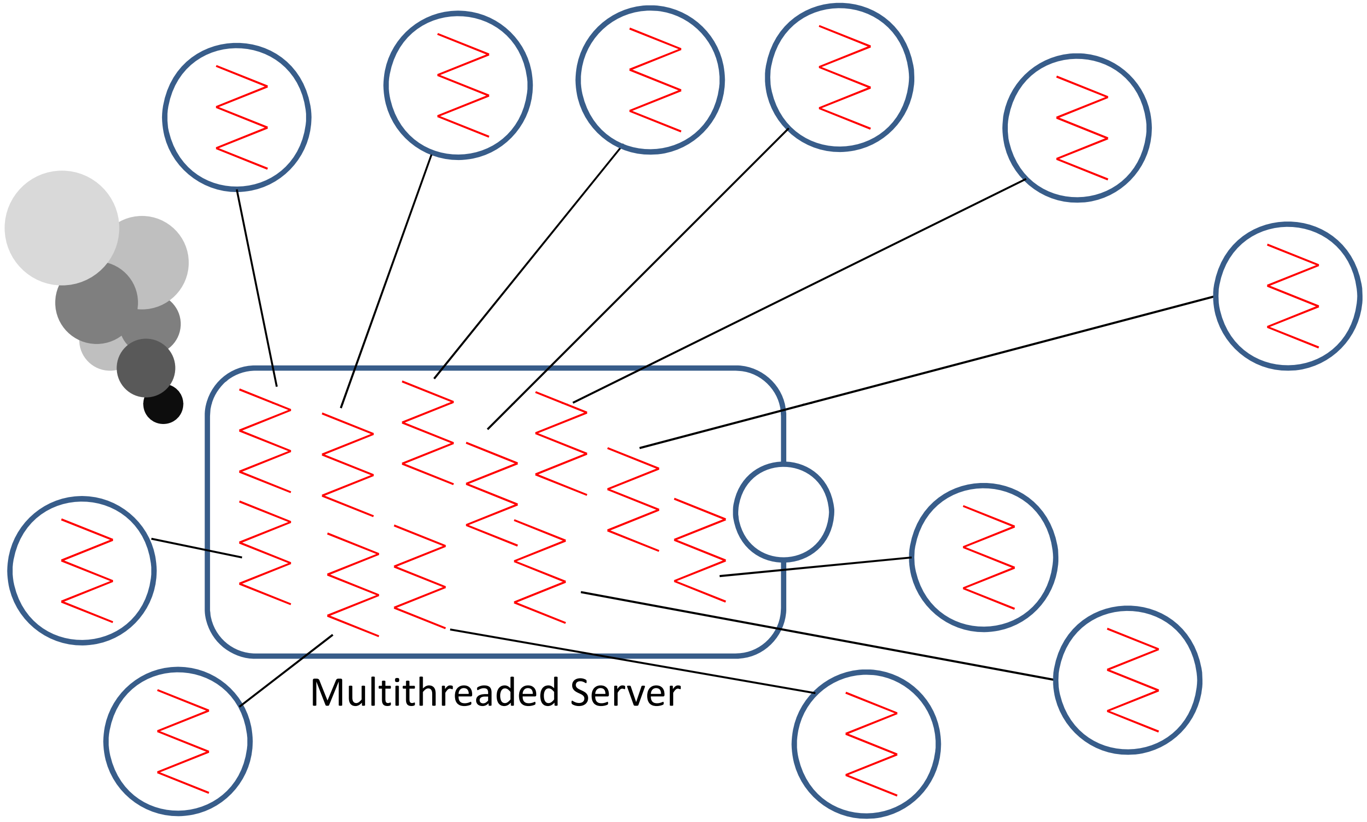
- … vs. multithreaded concurrency



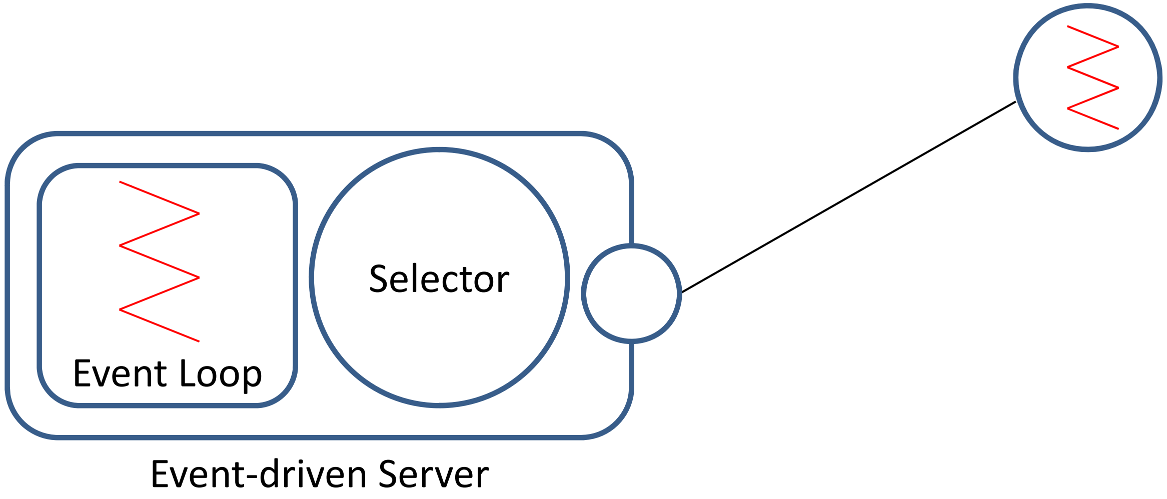
Multithreaded Server

Event-driven Concurrency

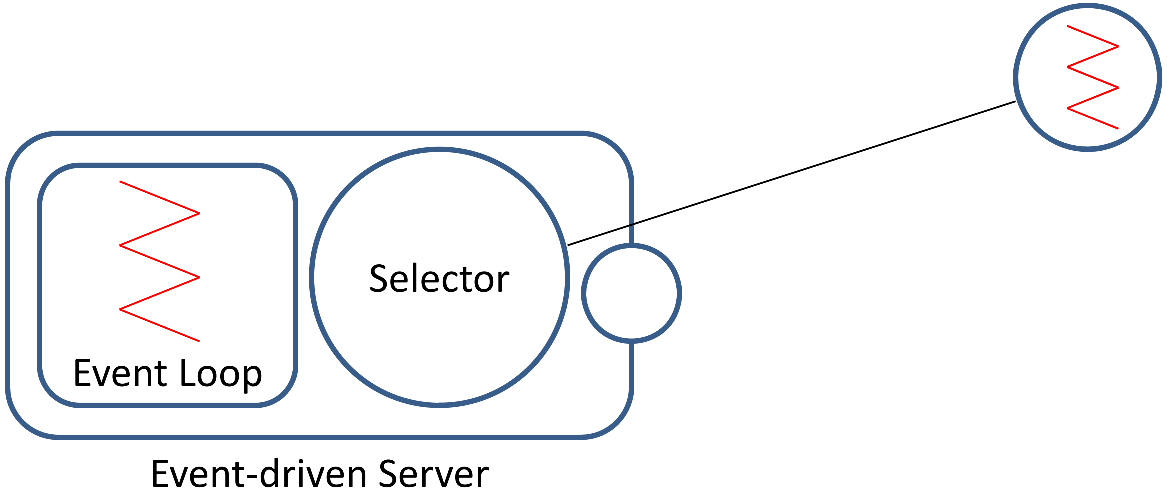
- … vs. multithreaded concurrency



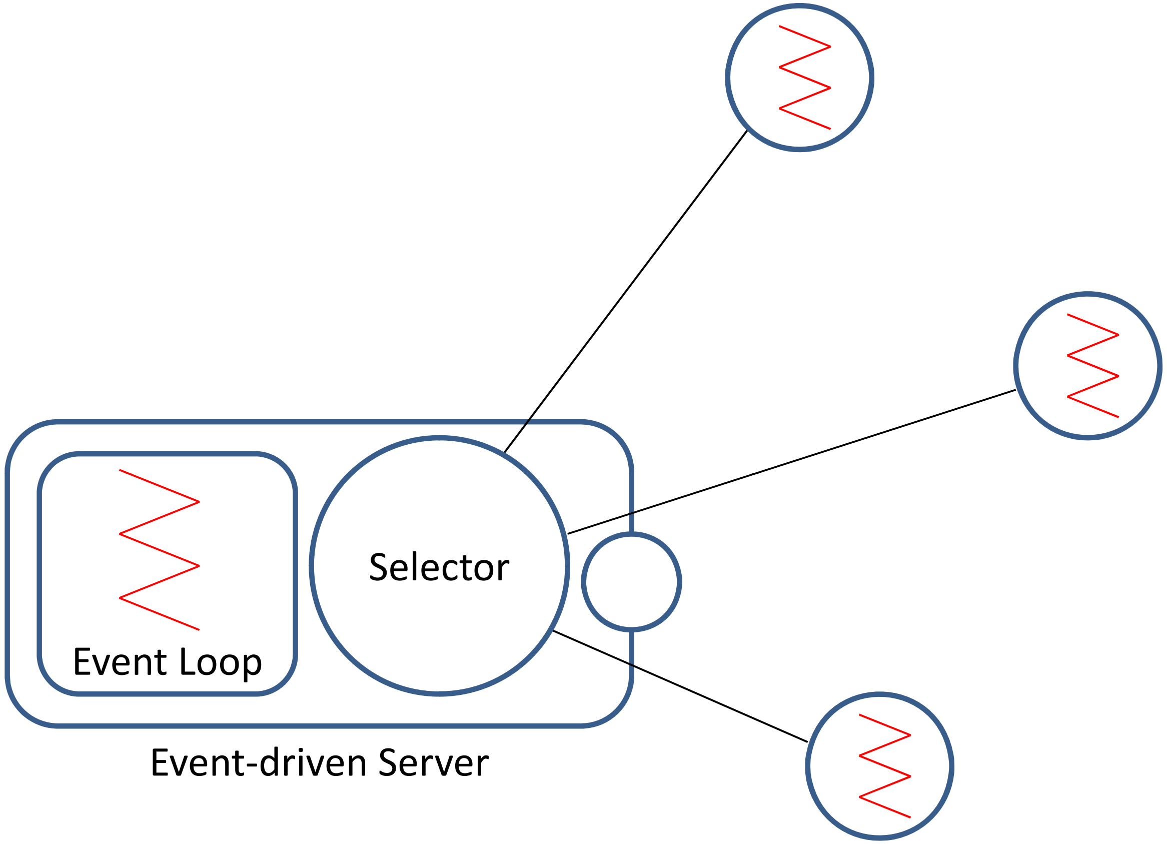
Event-driven Concurrency



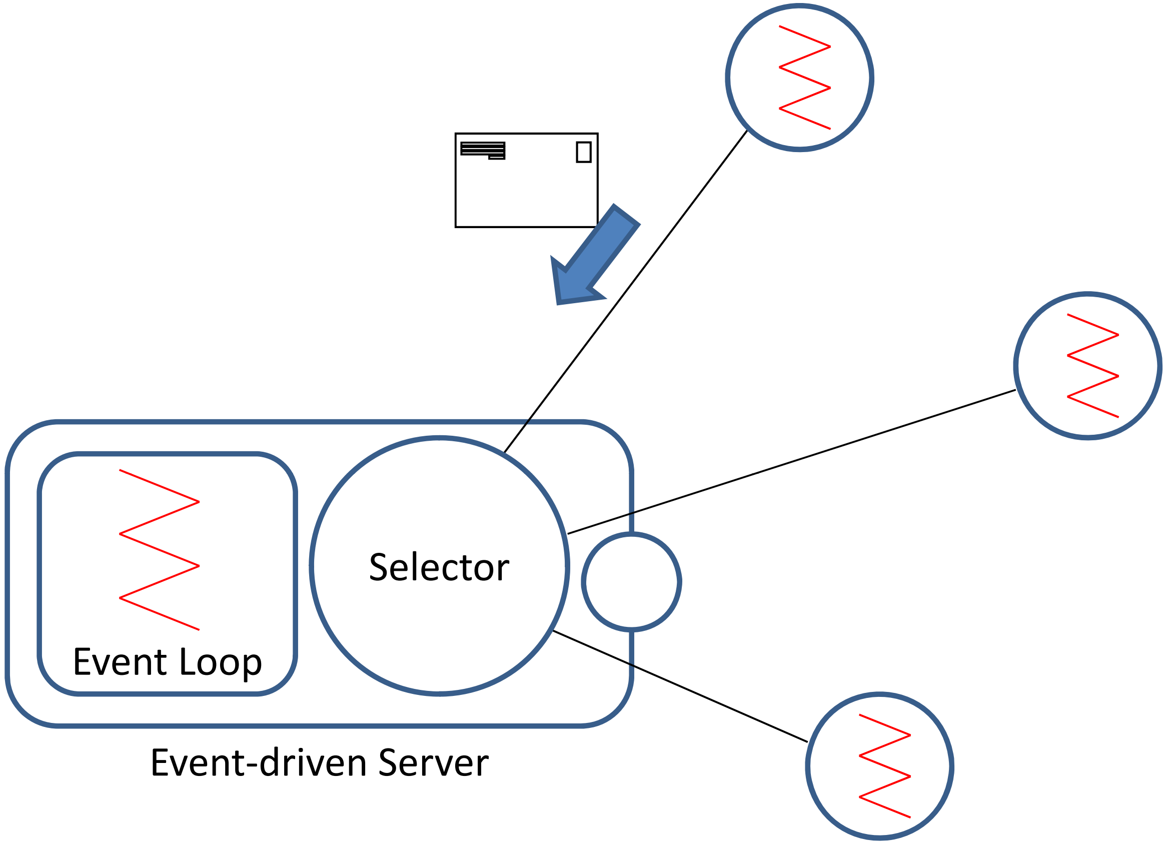
Event-driven Concurrency



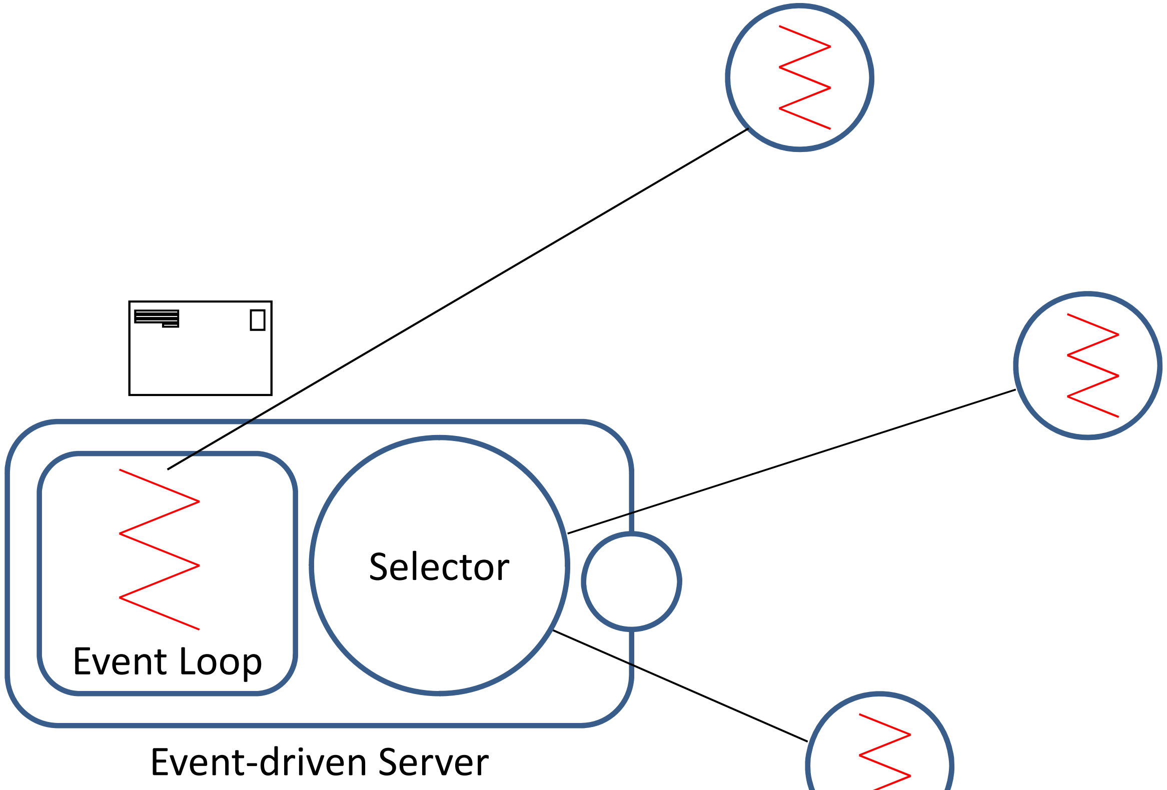
Event-driven Concurrency



Event-driven Concurrency

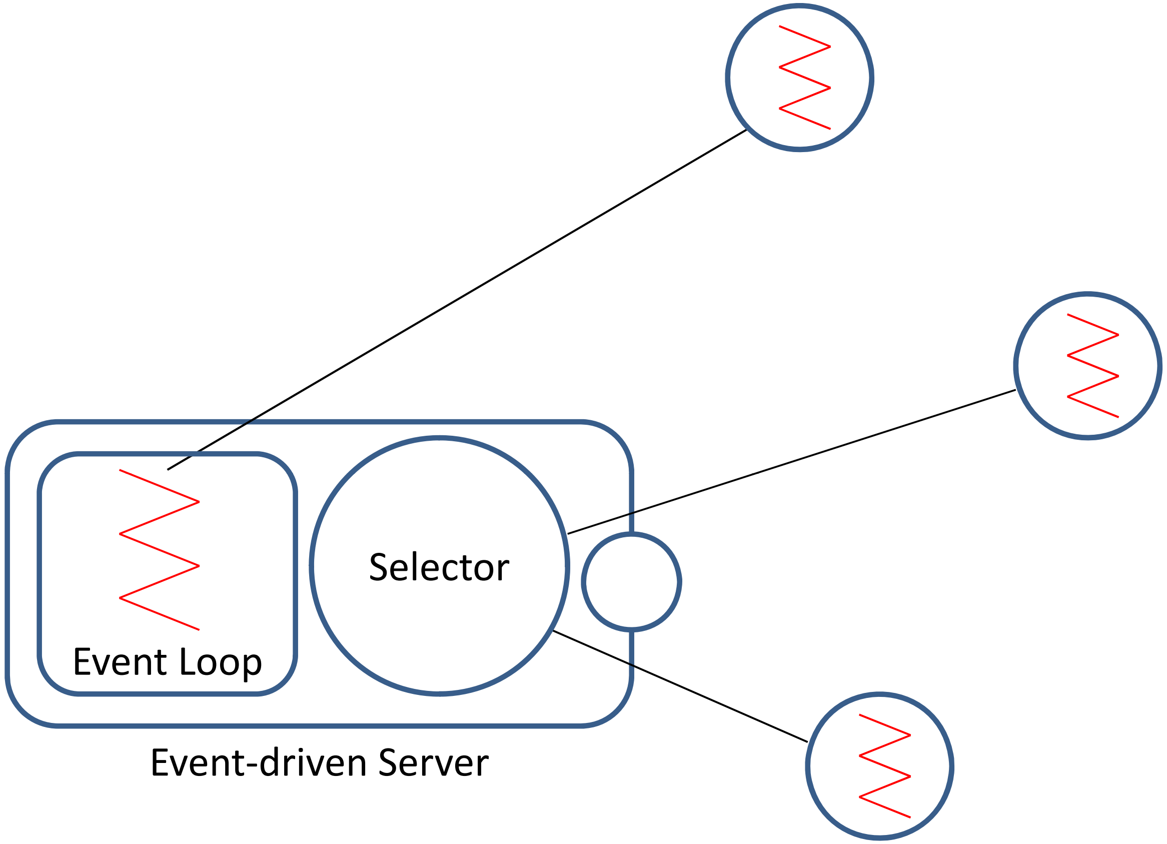


Event-driven Concurrency



Event-driven Concurrency

+ scalability

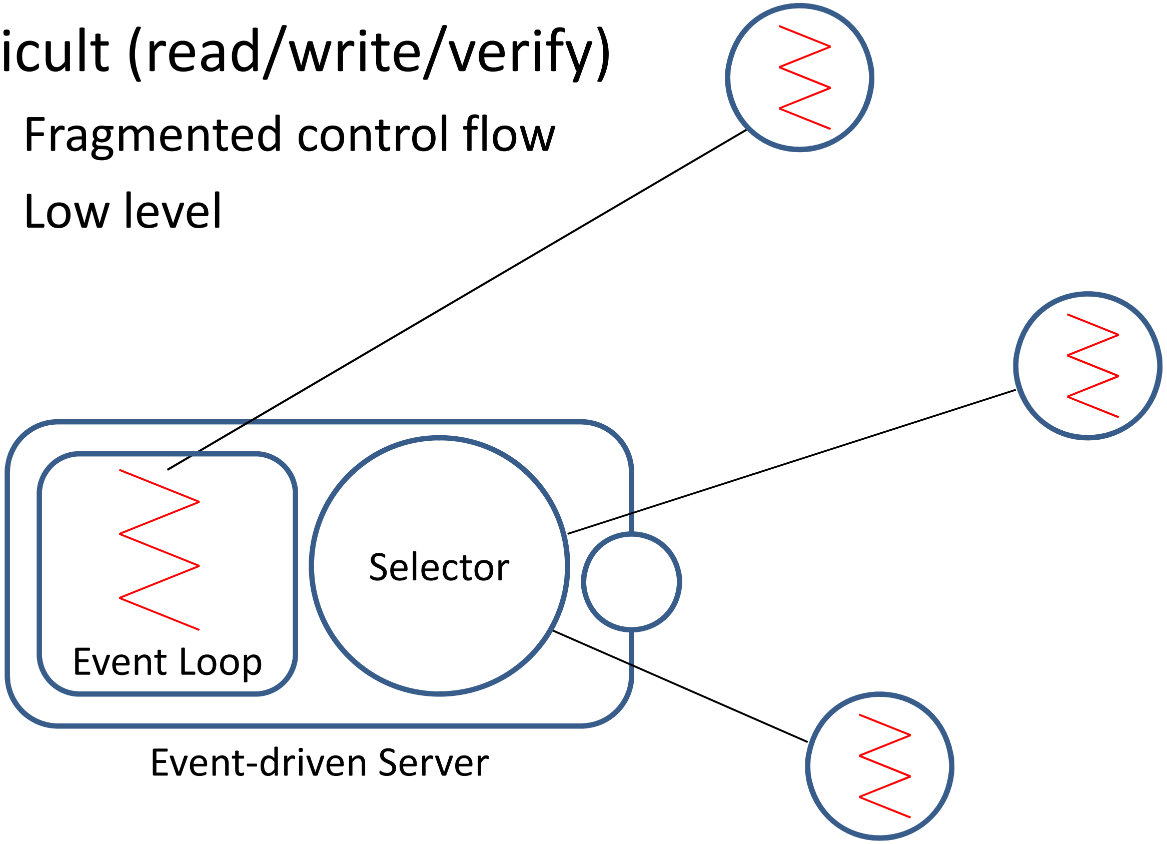


Event-driven Concurrency

+ scalability

– difficult (read/write/verify)

- Fragmented control flow
- Low level

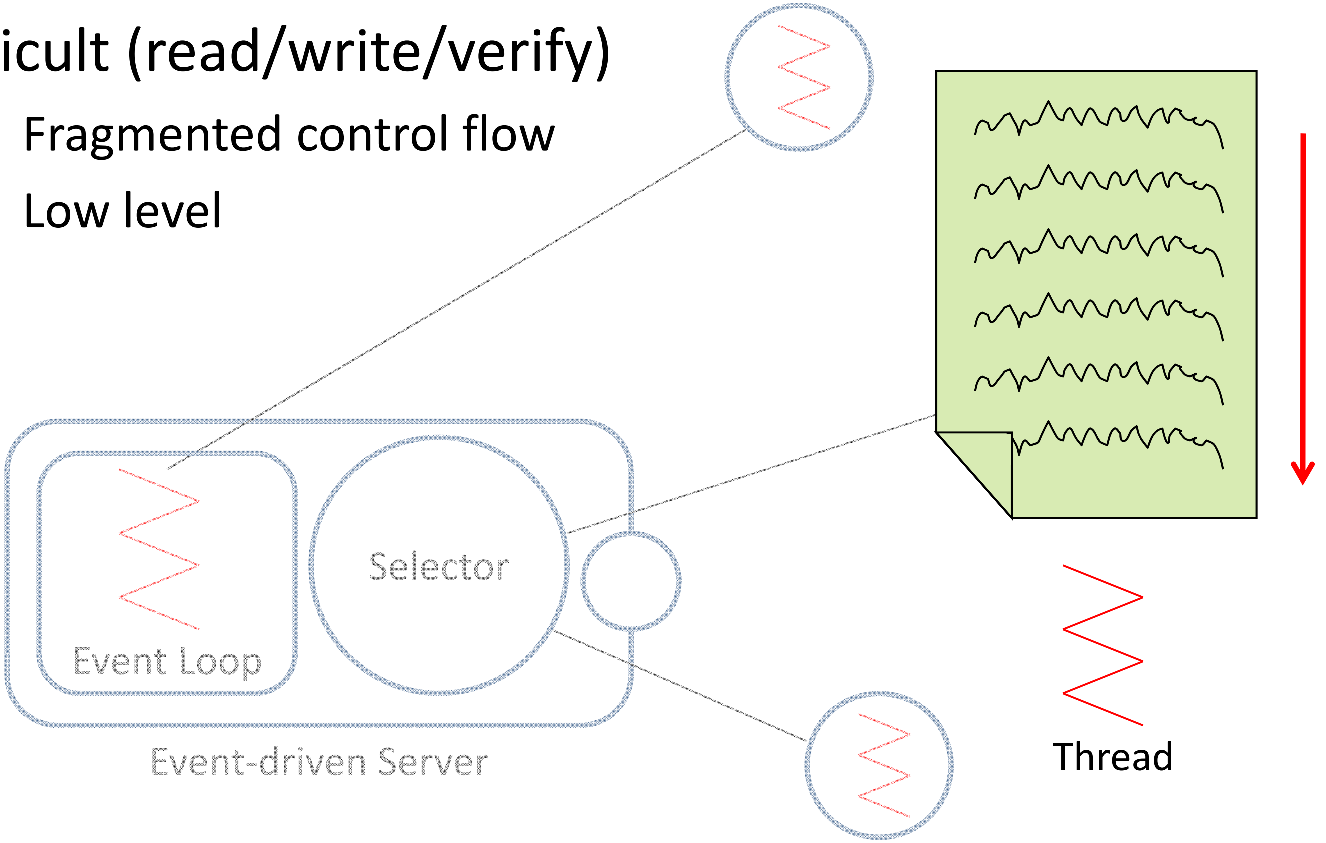


Event-driven Concurrency

+ scalability

– difficult (read/write/verify)

- Fragmented control flow
- Low level

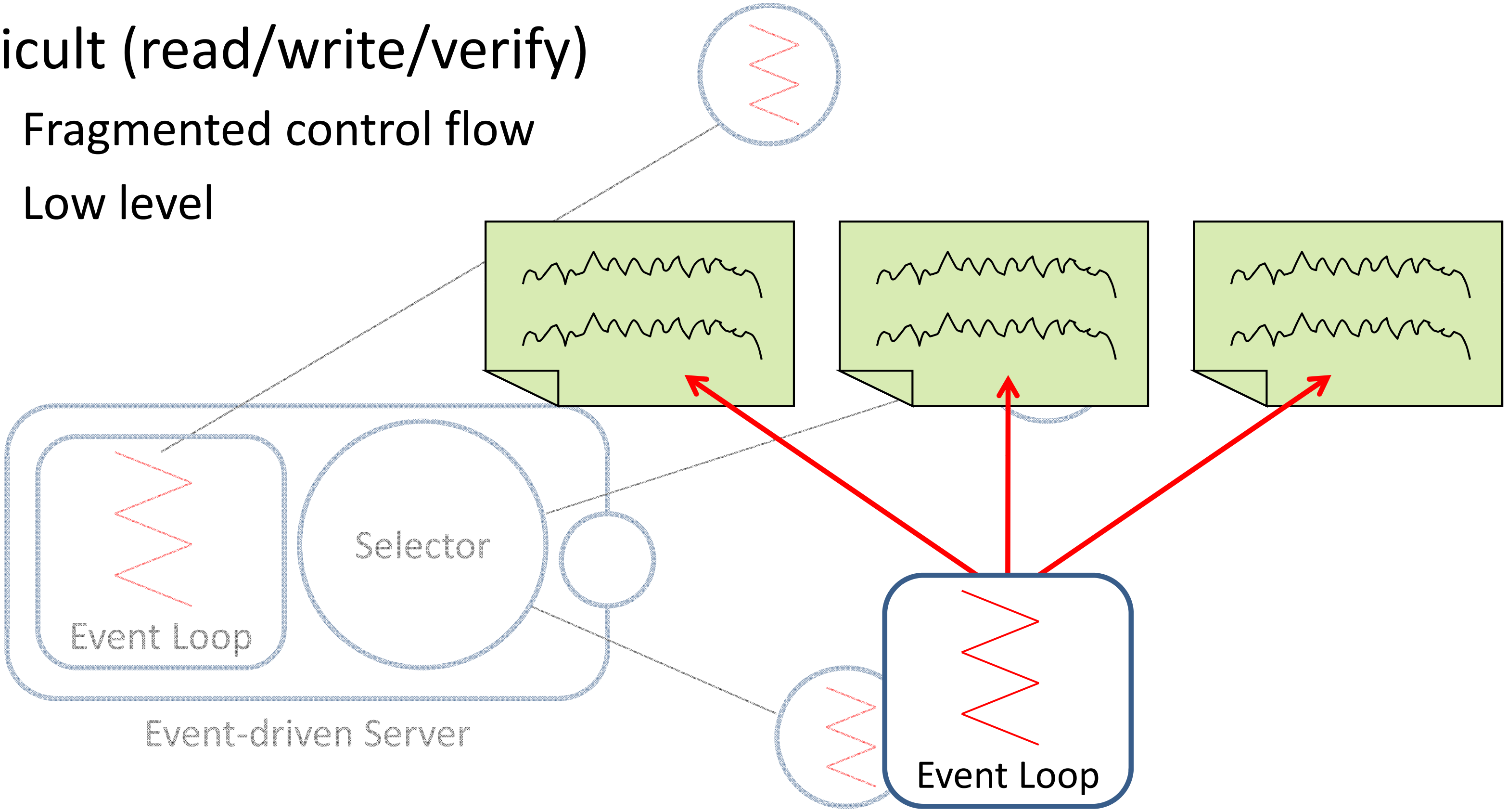


Event-driven Concurrency

+ scalability

– difficult (read/write/verify)

- Fragmented control flow
- Low level



Language and Runtime for Events

Language features for event-driven programming…

- ***Events can make Sense***. M. Krohn, E. Kohler, and M. F. Kaashoek. (*USENIX ATC 2007.)*
- ***EventJava: An Extension of Java for Event Correlation***. P. Eugster and K. R. Jayaram. (*ECOOP 2009.)*

Alternative programming interfaces over event-driven runtimes…

- ***Combining Events and Threads for Scalable Network Services***. P. Li and S. Zdancewic. (*PLDI 2007.)*
- ***Scala Actors: Unifying Thread-based and Event-based Programming***. P. Haller and M. Odersky. (*TCS 2009.)*
- ***Capriccio: Scalable Threads for Internet Services***. Capriccio R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. Brewer. (*SOSP 2003.)*

Session Types

Theory…

- ***Language Primitives and Type Disciplines for Structured Communication-based Programming***. K. Honda, V. T. Vasconcelos, and M. Kubo. (*ESOP 1998.)*
- ***Session types for Object-oriented Languages***. M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. (*ECOOP 2006*.)

Practical language design and implementations…

- ***Session-based Distributed Programming in Java***. R. Hu, N. Yoshida, and K. Honda. (*ECOOP 2008.)*
- ***Modular Session Types for Distributed Object-oriented Programming***. S. J. Gay, V. T. Vasconcelos, A. Ravara, N. Gesbert, and A. Z. Caldeira. (*POPL 2010*.)
- ***Language Support for Fast and Reliable Message-based Communication in Singularity OS***. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. Hunt, J. R. Larus, and S. Levi. (*EuroSys 2006.)*

Type-safe Eventful Sessions in Java

```
begin
.!< Background, Contributions >

.!< Basic Example >
.!< Formalism and Properties >
.!< Implementation, Real-world Example: SMTP >

.!< Conclusion >
.?[
        ?( Question ).!< Answer >
]*
.end
```

Session-typed Event Programming

- Handle concurrent sessions of type:

```
sbegin.?(Data).?(Data).!<Result>
```

Session-based communications programming in SJ:

- Declaration of communication protocols using session types
- Implementation of protocols using statically type-checked session operations

Session-typed Event Programming

- Handle concurrent sessions of type:

```
sbegin.?(Data).?(Data).!<Result>
```

Session-typed Event Programming

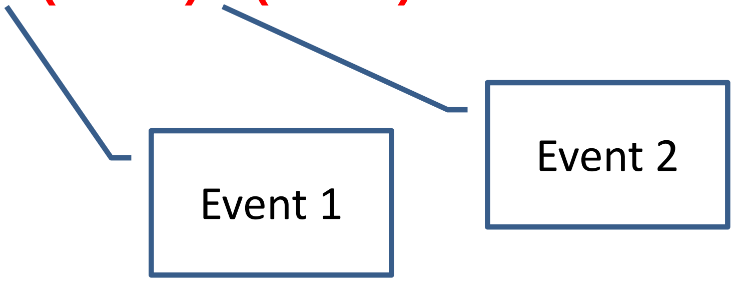
- Handle concurrent sessions of type:

```
protocol pServer {  sbegin.?(Data).?(Data).!<Result>  }
```

Session-typed Event Programming

- Handle concurrent sessions of type:

```
protocol pServer {  sbegin.?(Data).?(Data).!<Result>  }
```



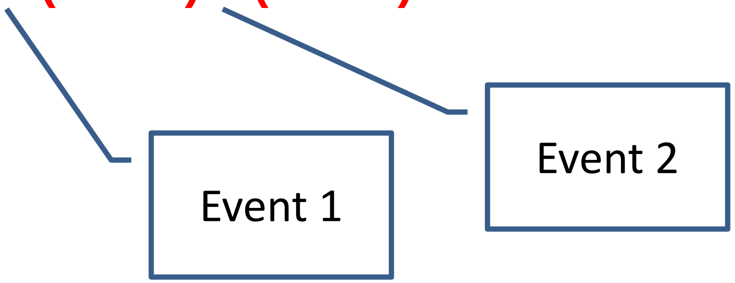
Event 1

Event 2

Session-typed Event Programming

- Handle concurrent sessions of type:

```
protocol pServer {  sbegin.?(Data).?(Data).!<Result>  }
```

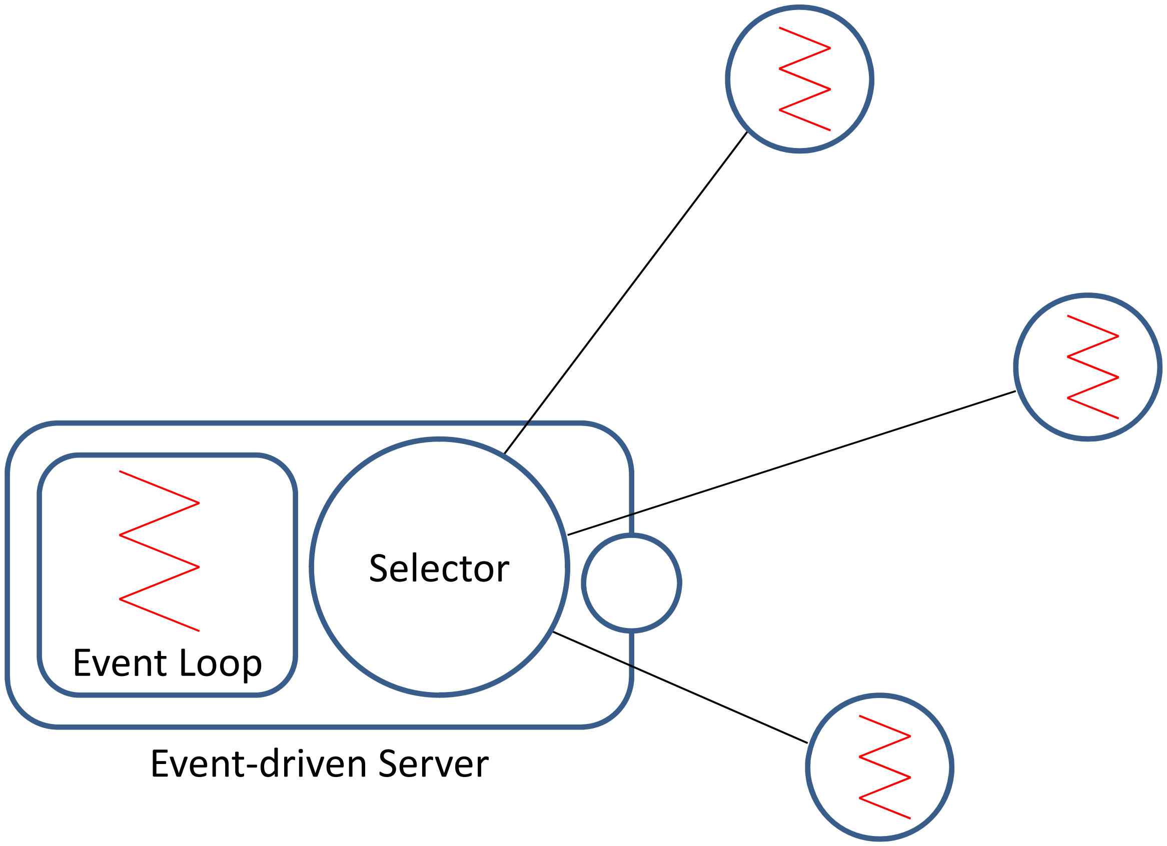


Event 1

Event 2

```
// Session set type
protocol pSelector {
  ?(Data).?(Data).!<Result>,      // Event 1
  ?(Data).!<Result>               // Event 2
}
```

Event-driven Concurrency



Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }










} }
```

Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
























} }
```

Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...



} }
```

Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...



} }
```

Session Socket endpoint:

`SJSocket{?(Data).?(Data).!<Result>}`

Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {






} } } }
```

Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) { // Event 1
            Data d1 = s1.receive();         // ?(Data)
            sel.register(s1);               // ?(Data).!<Result>
          }




} } } } }
```

Session-typed Event Programming

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) {
            Data d1 = s1.receive();
            sel.register(s1);
          }
          when(SJSocket{?(Data).!<Result>} s2) { // Event 2
            Data d2 = s2.receive();        // ?(Data)
            s2.send(new Result(...));      // !<Result>
          }
} } } } }
```

Session-typed Event Pr

Explicit specification of system events

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) {
            Data d1 = s1.receive();
            sel.register(s1);
          }
          when(SJSocket{?(Data).!<Result>} s2) {
            Data d2 = s2.receive();
            s2.send(new Result(...));
          }
} } } } }
```

Session-typed Event Pr

Explicit specification of system events

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) {
            Data d1 = s1.receive();
            sel.register(s1);
          }
          when(SJSocket{?(Data).!<Result>} s2) {
            Data d2 = s2.receive();
            s2.send(new Result(...));
          }
} } } } }
```

Precise specification of each event

Session-typed Event Pr

Explicit specification of system events

Precise specification of each event

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) {
            Data d1 = s1.receive();
            sel.register(s1);
          }
          when(SJSocket{?(Data).!<Result>} s2) {
            Data d2 = s2.receive();
            s2.send(new Result(...));
          }
} } } } }
```

Session-typed Event Pr

Explicit specification of system events

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) {
            Data d1 = s1.receive();
            sel.register(s1);
          }
          when(SJSocket{?(Data).!<Result>} s2) {
            Data d2 = s2.receive();
            s2.send(new Result(...));
          }
} } } } }
```

Precise specification of each event

Correct matching of events to event handlers

Session-typed Event Pr

Explicit specification of system events

```
class Example {
  public static void main(String[] args) throws SJIOException {
    protocol pSelector { ?(Data).?(Data).!<Result>, ?(Data).!<Result> }
    SJSelector{pSelector} sel = new SJSelector{pSelector}();
    ...
    sel.register(client);
    ...
    while(run) { // Main event loop.
      using(SJSocket{pSelector} s = sel.select()) {
        typecase(s) {
          when(SJSocket{?(Data).?(Data).!<Result>} s1) {
            Data d1 = s1.receive();
            sel.register(s1);
          }
          when(SJSocket{?(Data).!<Result>} s2) {
            Data d2 = s2.receive();
            s2.send(new Result(...));
          }
} } } } }
```

Precise specification of each event

Correct matching of events to event handlers

Correct handling of each event; preservation of session flow across event boundaries

Type-safe Eventful Sessions in Java

```
begin
.!< Background, Contributions >
.!< Basic Example >

.!< Formalism and Properties >
.!< Implementation, Real-world Example: SMTP >

.!< Conclusion >
.?[
        ?( Question ).!< Answer >
]*
.end
```

Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida, Kohei Honda

Eventful Session Pi-calculus (ESP)

- To model *session-typed event-driven programming*, we extend the standard session pi-calculus with:
  - Message **`arrived`** primitive: minimal mechanism for asynchronous (non-blocking) input
  - Session **`typecase`**: dynamic inspection of session types
  - Session *set types*: heterogeneous session type collections

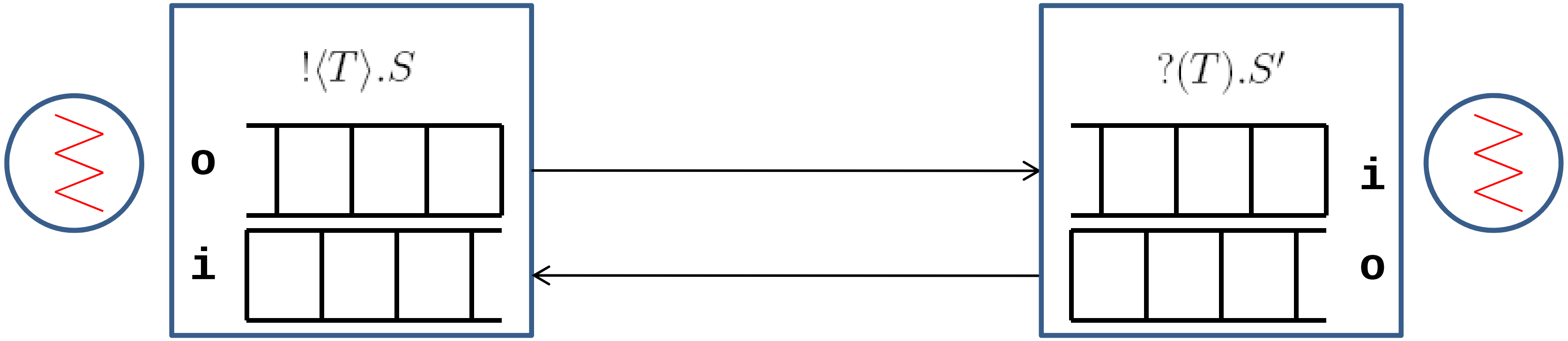
Eventful Session Pi-calculus (ESP)

- To model *session-typed event-driven programming*, we extend the standard session pi-calculus with:
  - Message **`arrived`** primitive: minimal mechanism for asynchronous (non-blocking) input
  - Session **`typecase`**: dynamic inspection of session types
  - Session *set types*: heterogeneous session type collections

➢ Encode higher-level event programming constructs using these primitives

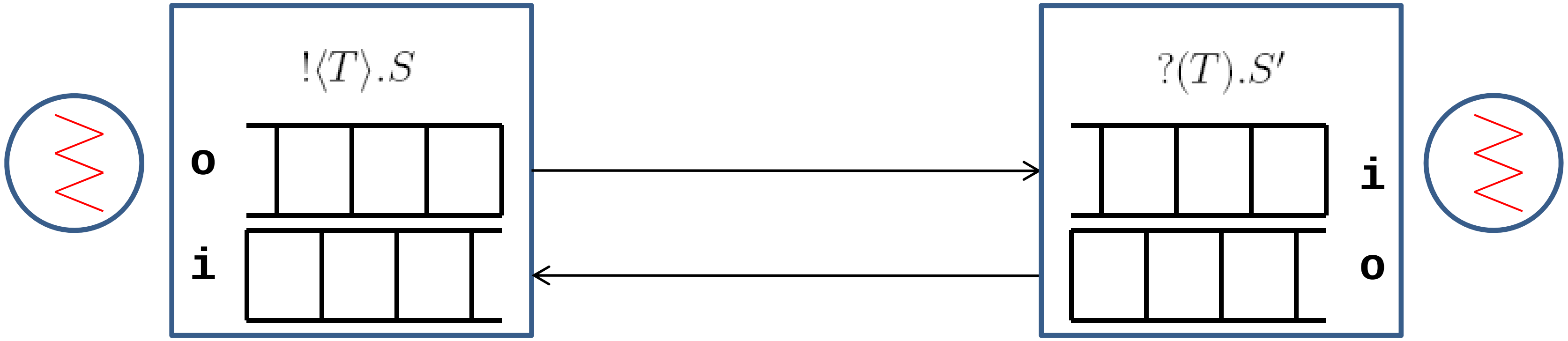
Operational Semantics

$s!\langle \mathbf{v} \rangle.P \mid s[!\langle T \rangle.S, \mathtt{i} : \varepsilon, \mathtt{o} : \varepsilon] \quad \mid \quad \overline{s}[?(T).S', \mathtt{i} : v, \mathtt{o} : \varepsilon] \mid \overline{s}?(x).Q$



Operational Semantics

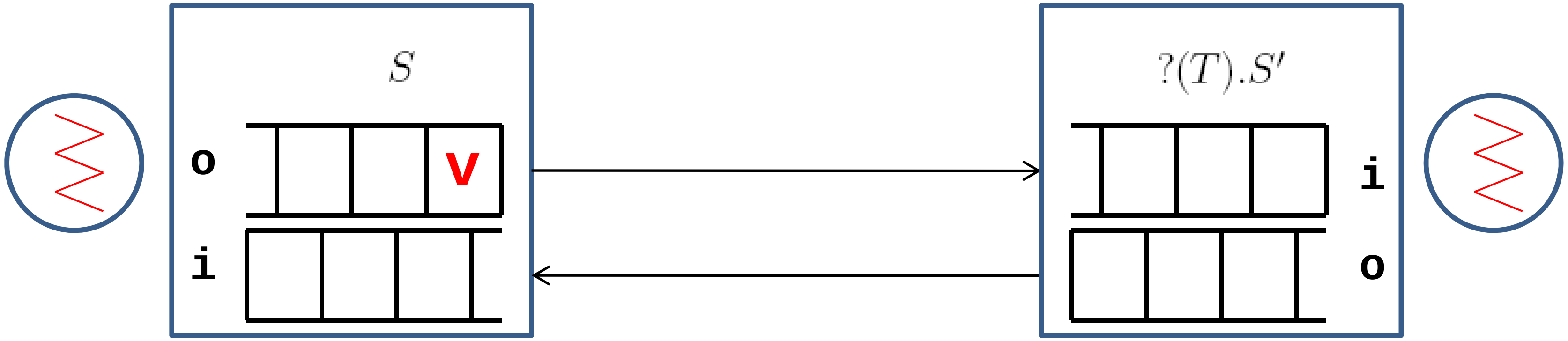
$s!\langle \mathbf{v}\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i}: \varepsilon, \mathtt{o}: \varepsilon]$ $\mid$ $\overline{s}[?(T).S', \mathtt{i}: v, \mathtt{o}: \varepsilon] \mid \overline{s}?(x).Q$



$$\frac{}{s!\langle v\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'] \longrightarrow P \mid s[S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'\cdot v]}\,[\text{Send}]$$

Operational Semantics

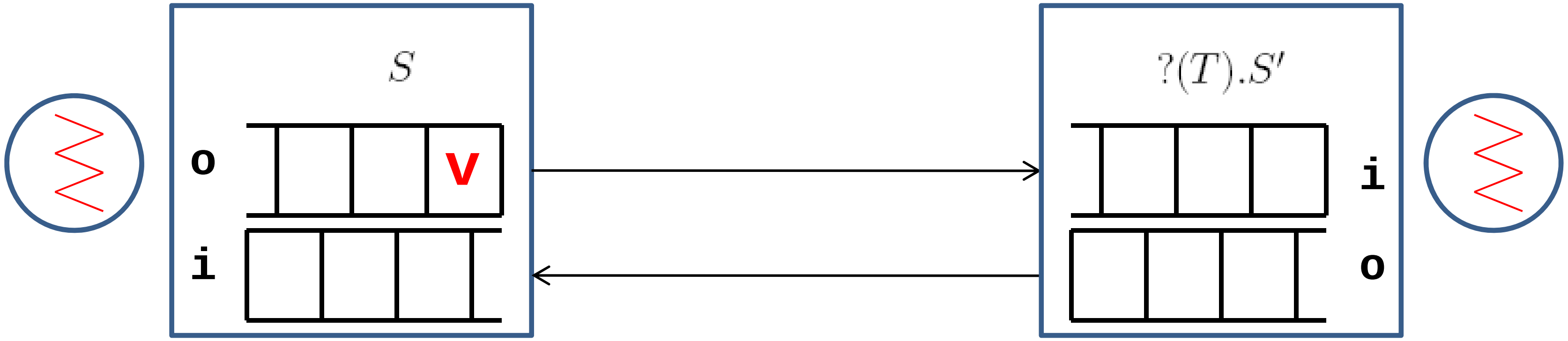
$P \mid s[S, \mathtt{i}: \varepsilon, \mathtt{o}: \mathrm{v}] \quad \mid \quad \overline{s}[?(T).S', \mathtt{i}: v, \mathtt{o}: \varepsilon] \mid \overline{s}?(x).Q$



$$\frac{}{s!\langle v\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'] \longrightarrow P \mid s[S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'\cdot v]}\ [\mathrm{Send}]$$

Operational Semantics

$P \mid s[S, \mathtt{i}\colon \varepsilon, \mathtt{o}\colon \mathrm{v}] \qquad \mid \qquad \overline{s}[?(T).S', \mathtt{i}\colon v, \mathtt{o}\colon \varepsilon] \mid \overline{s}?(x).Q$

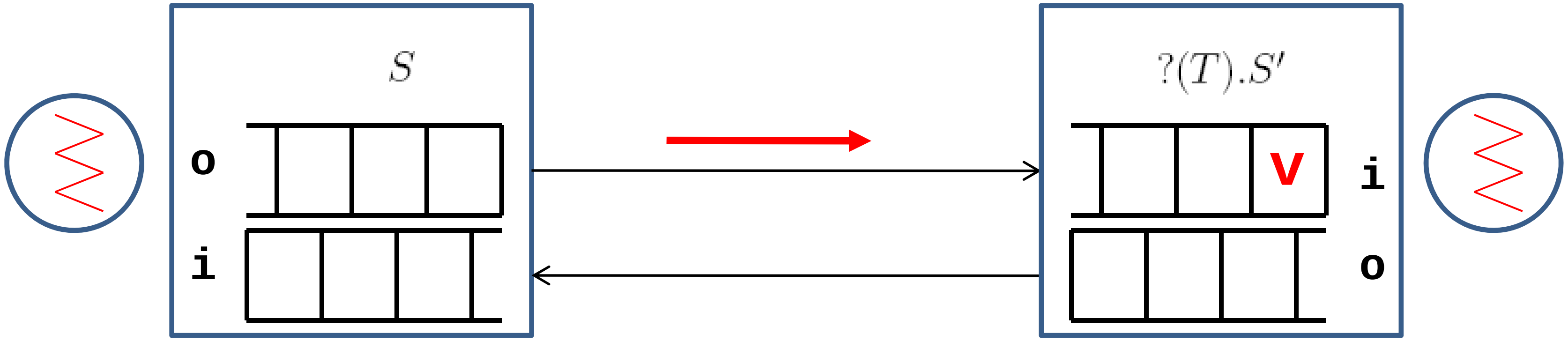


$$\frac{}{s!\langle v\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i}\colon \vec{h}, \mathtt{o}\colon \vec{h}'] \longrightarrow P \mid s[S, \mathtt{i}\colon \vec{h}, \mathtt{o}\colon \vec{h}'\cdot v]}\ [\mathrm{Send}]$$

$$\frac{}{s[\mathtt{o}\colon v\cdot\vec{h}] \mid \overline{s}[\mathtt{i}\colon \vec{h}'] \longrightarrow s[\mathtt{o}\colon \vec{h}] \mid \overline{s}[\mathtt{i}\colon \vec{h}'\cdot v]}\ [\mathrm{Comm}]$$

Operational Semantics

$P \mid s[S, \mathtt{i}: \varepsilon, \mathtt{o}: \varepsilon] \quad \mid \quad \overline{s}[?(T).S', \mathtt{i}: \mathtt{v}, \mathtt{o}: \varepsilon] \mid \overline{s}?(x).Q$

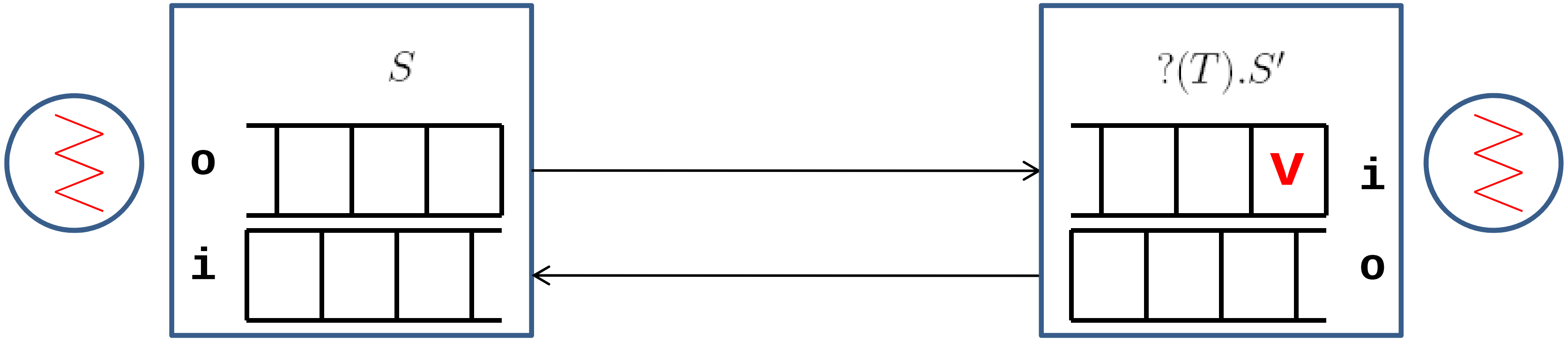


$$\frac{}{s!\langle v\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'] \longrightarrow P \mid s[S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'\cdot v]}\ [\text{Send}]$$

$$\frac{}{s[\mathtt{o}: v\cdot\vec{h}] \mid \overline{s}[\mathtt{i}: \vec{h}'] \longrightarrow s[\mathtt{o}: \vec{h}] \mid \overline{s}[\mathtt{i}: \vec{h}'\cdot v]}\ [\text{Comm}]$$

Operational Semantics

$P \mid s[S, \mathtt{i} \colon \varepsilon, \mathtt{o} \colon \varepsilon] \quad \mid \quad \overline{s}[?(T).S', \mathtt{i} \colon \mathtt{v}, \mathtt{o} \colon \varepsilon] \mid \overline{s}?(x).Q$



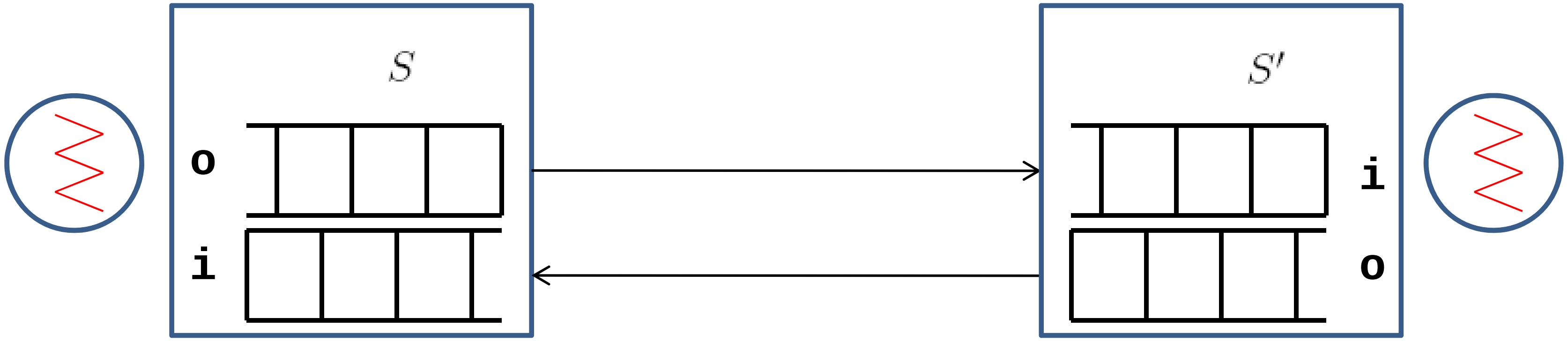
$$\frac{}{s!\langle v\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i} \colon \vec{h}, \mathtt{o} \colon \vec{h}'] \longrightarrow P \mid s[S, \mathtt{i} \colon \vec{h}, \mathtt{o} \colon \vec{h}'\cdot v]}\ [\text{Send}]$$

$$\frac{}{s[\mathtt{o} \colon v\cdot\vec{h}] \mid \overline{s}[\mathtt{i} \colon \vec{h}'] \longrightarrow s[\mathtt{o} \colon \vec{h}] \mid \overline{s}[\mathtt{i} \colon \vec{h}'\cdot v]}\ [\text{Comm}]$$

$$\frac{}{s?(x).P \mid s[?(T).S, \mathtt{i} \colon v\cdot\vec{h}, \mathtt{o} \colon \vec{h}'] \longrightarrow P\{v/x\} \mid s[S, \mathtt{i} \colon \vec{h}, \mathtt{o} \colon \vec{h}']}\ [\text{Receive}]$$

Operational Semantics

$P \mid s[S, \mathtt{i}: \varepsilon, \mathtt{o}: \varepsilon] \quad \mid \quad \overline{s}[S', \mathtt{i}: \varepsilon, \mathtt{o}: \varepsilon] \mid Q\{\mathbf{v}/x\}$

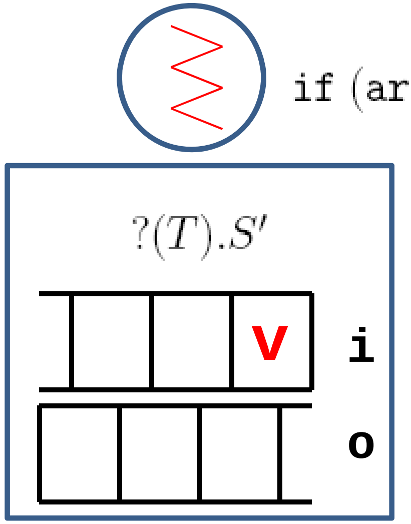


$$\frac{}{s!\langle v\rangle.P \mid s[!\langle T\rangle.S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'] \longrightarrow P \mid s[S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}'\cdot v]}\ [\text{Send}]$$

$$\frac{}{s[\mathtt{o}: v\cdot\vec{h}] \mid \overline{s}[\mathtt{i}: \vec{h}'] \longrightarrow s[\mathtt{o}: \vec{h}] \mid \overline{s}[\mathtt{i}: \vec{h}'\cdot v]}\ [\text{Comm}]$$

$$\frac{}{s?(x).P \mid s[?(T).S, \mathtt{i}: v\cdot\vec{h}, \mathtt{o}: \vec{h}'] \longrightarrow P\{v/x\} \mid s[S, \mathtt{i}: \vec{h}, \mathtt{o}: \vec{h}']}\ [\text{Receive}]$$

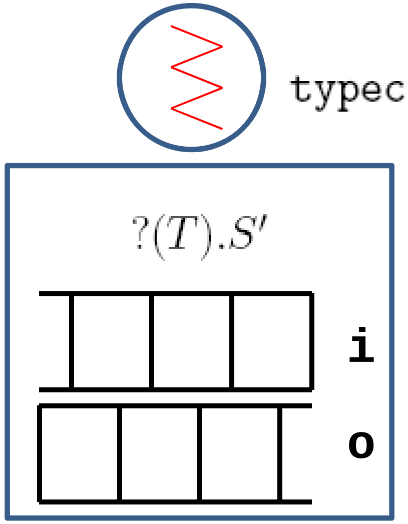
Operational Semantics



$\texttt{if}\ (\texttt{arrived}\ s)\ \texttt{then}\ P\ \texttt{else}\ Q$

$$\frac{(\vec{h} \neq \varepsilon) \downarrow b}{E[\texttt{arrived}\ s] \mid s[S, \texttt{i}: \vec{h}, \texttt{o}: \vec{h}'] \longrightarrow E[b] \mid s[S, \texttt{i}: \vec{h}, \texttt{o}: \vec{h}']}\ [\text{Arrived}]$$

Operational Semantics

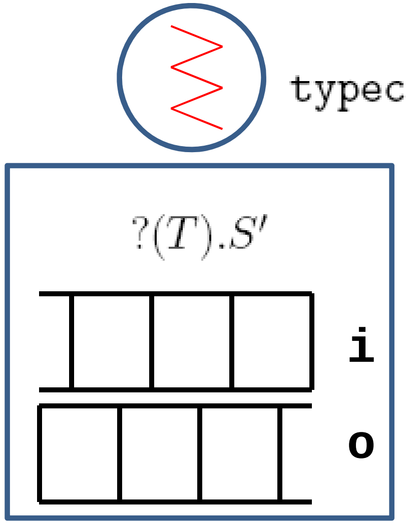


$\texttt{typecase}\ s\ \{(x_1 : T_1)\ P_1, \ldots, (x_n : T_n)\ P_n\}$

$$\frac{(\vec{h} \neq \varepsilon) \downarrow b}{E[\texttt{arrived}\ s] \mid s[S, \texttt{i}: \vec{h}, \texttt{o}: \vec{h}'] \longrightarrow E[b] \mid s[S, \texttt{i}: \vec{h}, \texttt{o}: \vec{h}']}\ [\text{Arrived}]$$

$$\frac{\forall j < i. T_j \not\leq S \wedge T_i \leq S}{\texttt{typecase}\ s\ \{(x_i : T_i)\ P_i\}_{i \in I} \mid s[S] \longrightarrow P_i\{s/x_i\} \mid s[S]}\ [\text{Typecase}]$$

Operational Semantics

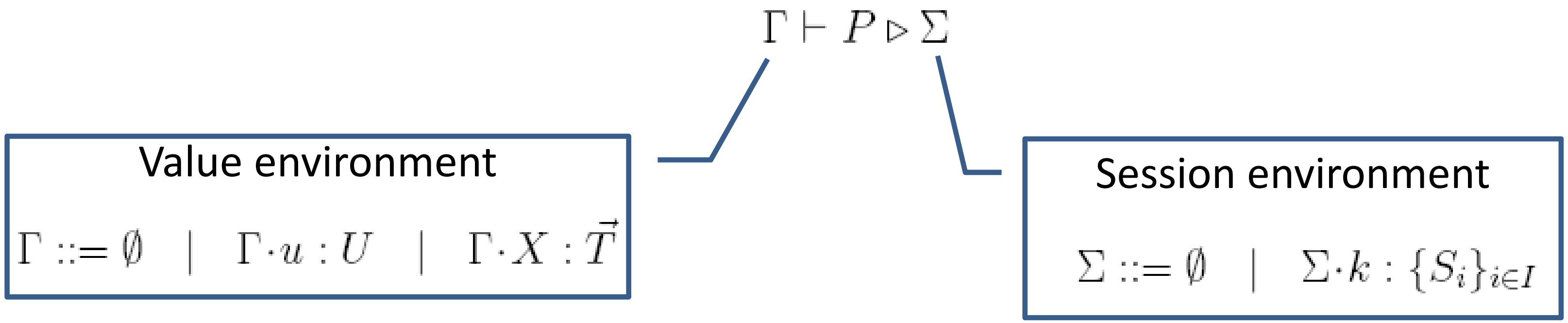


$$\mathtt{typecase}\ s\ \{(x_1 : T_1)\ P_1, \ldots, (x_n : T_n)\ P_n\} \longrightarrow P_i\{s/x_i\}$$

$$\frac{(\vec{h} \neq \varepsilon) \downarrow b}{E[\mathtt{arrived}\ s] \mid s[S, \mathtt{i} : \vec{h}, \mathtt{o} : \vec{h}'] \longrightarrow E[b] \mid s[S, \mathtt{i} : \vec{h}, \mathtt{o} : \vec{h}']}\ [\text{Arrived}]$$

$$\frac{\forall j < i. T_j \not\leq S \wedge T_i \leq S}{\mathtt{typecase}\ s\ \{(x_i : T_i)\ P_i\}_{i \in I} \mid s[S] \longrightarrow P_i\{s/x_i\} \mid s[S]}\ [\text{Typecase}]$$

Typing and Communication Safety



$$\Gamma \vdash P \triangleright \Sigma$$

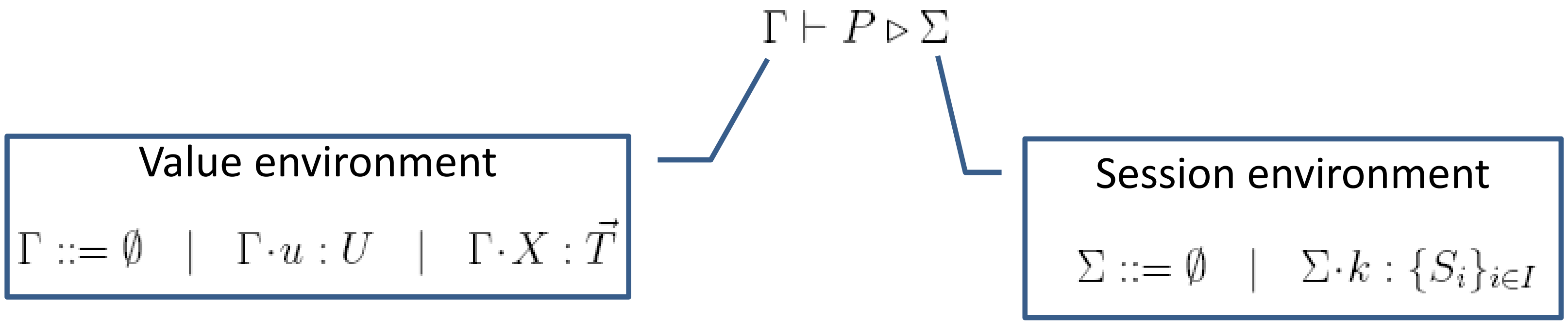
Value environment

$$\Gamma ::= \emptyset \quad | \quad \Gamma\cdot u : U \quad | \quad \Gamma\cdot X : \vec{T}$$

Session environment

$$\Sigma ::= \emptyset \quad | \quad \Sigma\cdot k : \{S_i\}_{i\in I}$$

Typing and Communication Safety



$$\Gamma \vdash P \triangleright \Sigma$$

Value environment

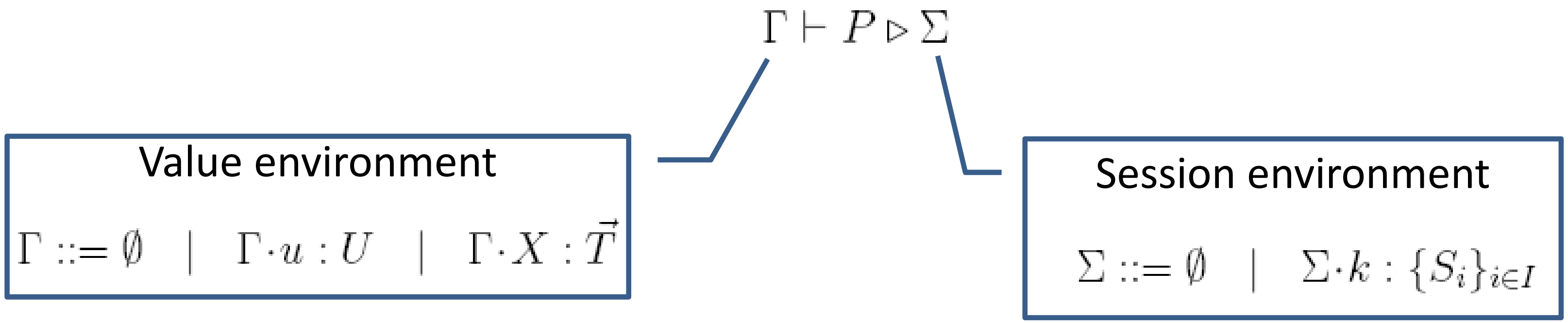
$$\Gamma ::= \emptyset \quad \mid \quad \Gamma\cdot u : U \quad \mid \quad \Gamma\cdot X : \vec{T}$$

Session environment

$$\Sigma ::= \emptyset \quad \mid \quad \Sigma\cdot k : \{S_i\}_{i\in I}$$

$$\frac{\Gamma, \Sigma \vdash e : U \quad \Gamma \vdash P \triangleright \Sigma\cdot k : S}{\Gamma \vdash k!\langle e\rangle.P \triangleright \Sigma\cdot k : !\langle U\rangle.S}\ (\mathrm{Send})$$

Typing and Communication Safety



$$\Gamma \vdash P \triangleright \Sigma$$

Value environment

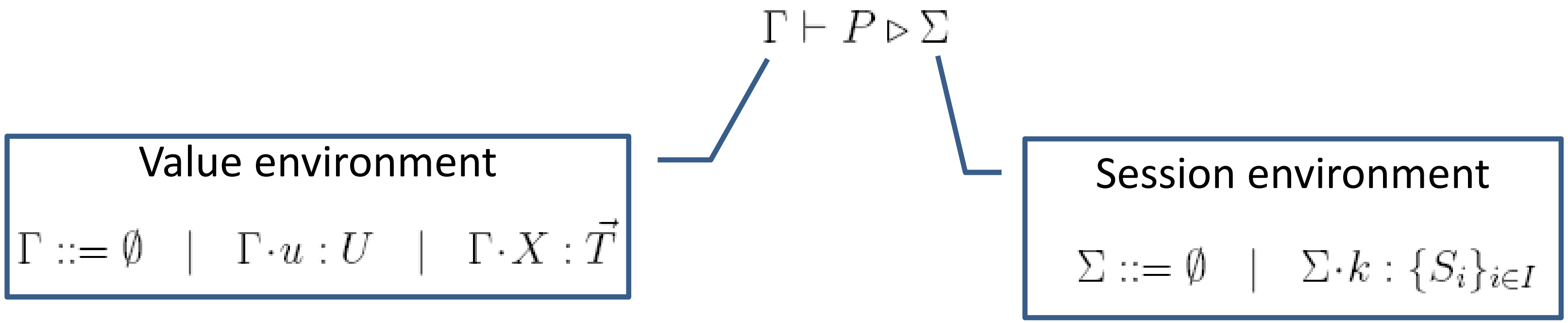
$$\Gamma ::= \emptyset \quad | \quad \Gamma\cdot u : U \quad | \quad \Gamma\cdot X : \vec{T}$$

Session environment

$$\Sigma ::= \emptyset \quad | \quad \Sigma\cdot k : \{S_i\}_{i\in I}$$

$$\frac{\forall i \in I.\Gamma \vdash P_i \triangleright \Sigma\cdot x_i : S_i}{\Gamma \vdash \mathtt{typecase}\; k\; \{(x_i : S_i)\; P_i\}_{i\in I} \triangleright \Sigma\cdot k : \{S_i\}_{i\in I}}\;(\mathrm{Typecase})$$

Typing and Communication Safety



$\Gamma \vdash P \triangleright \Sigma$

Value environment

$\Gamma ::= \emptyset \quad | \quad \Gamma \cdot u : U \quad | \quad \Gamma \cdot X : \vec{T}$

Session environment

$\Sigma ::= \emptyset \quad | \quad \Sigma \cdot k : \{S_i\}_{i \in I}$

$$\frac{\forall i \in I. \Gamma \vdash P_i \triangleright \Sigma \cdot x_i : S_i}{\Gamma \vdash \mathtt{typecase}\ k\ \{(x_i : S_i)\ P_i\}_{i \in I} \triangleright \Sigma \cdot k : \{S_i\}_{i \in I}}\ (\text{Typecase})$$

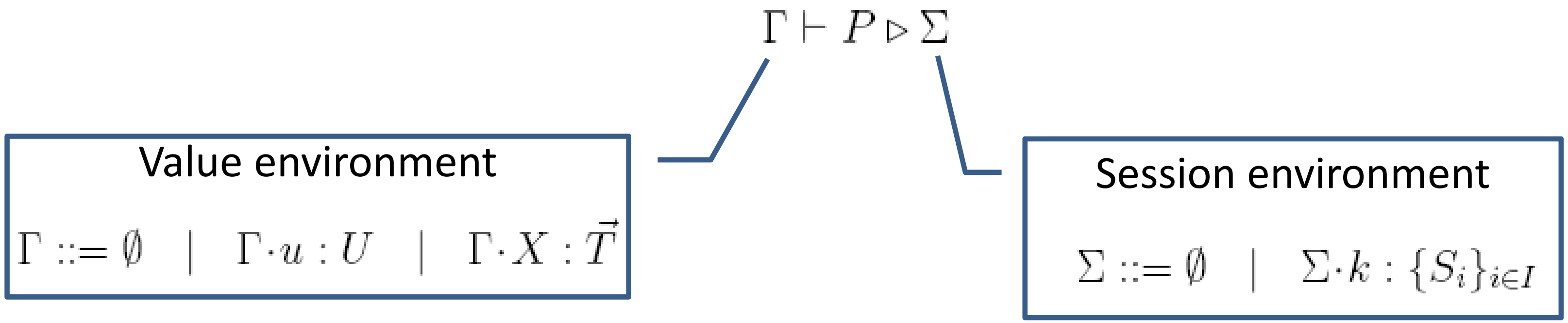
Theorem. (Subject Reduction)

If $\Gamma \vdash P \triangleright \emptyset$ *and* $P \longrightarrow Q$, *then we have* $\Gamma \vdash Q \triangleright \emptyset$.

Theorem. (Communication Safety)

If $\Gamma \vdash P \triangleright \emptyset$, *then* P *never reduces to an error.*

Typing and Communication Safety



$$\Gamma \vdash P \triangleright \Sigma$$

Value environment

$$\Gamma ::= \emptyset \quad | \quad \Gamma \cdot u : U \quad | \quad \Gamma \cdot X : \vec{T}$$

Session environment

$$\Sigma ::= \emptyset \quad | \quad \Sigma \cdot k : \{S_i\}_{i \in I}$$

$$\frac{\forall i \in I. \Gamma \vdash P_i \triangleright \Sigma \cdot x_i : S_i}{\Gamma \vdash \mathtt{typecase}\; k \; \{(x_i : S_i)\, P_i\}_{i \in I} \triangleright \Sigma \cdot k : \{S_i\}_{i \in I}} \; (\text{Typecase})$$

Event-handling Safety and Progress

- Represent higher-level event programming constructs using ESP
  - Selectors and event loops
  - Join patterns for correlating multiple events

Representing Selectors in ESP

ESP^{+}

ESP

Representing Selectors in ESP

ESP$^+$

```
SJSelector{pSelector} sel = new SJSelector{pSelector}();
sel.register(source);
while(run) { // Main event loop.
  using(SJSocket{pSelector} s = sel.select()) {
    typecase(s) {
      when(SJSocket{?(D).?(D).!<R>} s1) { …; sel.register(s1); }
      when(SJSocket{?(D).!<R>} s2) { … }
} } }
```

ESP

Representing Selectors in ESP

ESP+

```
SJSelector{pSelector} sel = new SJSelector{pSelector}();
sel.register(source);
while(run) { // Main event loop.
  using(SJSocket{pSelector} s = sel.select()) {
    typecase(s) {
      when(SJSocket{?(D).?(D).!<R>} s1) { …; sel.register(s1); }
      when(SJSocket{?(D).!<R>} s2) { … }
} } }
```

ESP $(\nu b)\,(b[\varepsilon] \mid \overline{b}(\overline{sel}).b(sel))$.

$)$

Representing Selectors in ESP

ESP+

```
SJSelector{pSelector} sel = new SJSelector{pSelector}();
sel.register(source);
while(run) { // Main event loop.
  using(SJSocket{pSelector} s = sel.select()) {
    typecase(s) {
      when(SJSocket{?(D).?(D).!<R>} s1) { …; sel.register(s1); }
      when(SJSocket{?(D).!<R>} s2) { … }
} } }
```

ESP

$(\nu b)\,(b[\varepsilon] \mid \overline{b}($[illegible]$).b(sel).$

$\overline{sel}!\langle s_1\rangle \ldots . \overline{sel}!\langle s_n\rangle.$

$.\overline{x}!\langle s_1\rangle$

$\}$

$\rangle)$

Representing Selectors in ESP

ESP$^+$

```
SJSelector{pSelector} sel = new SJSelector{pSelector}();
sel.register(source);
while(run) { // Main event loop.
  using(SJSocket{pSelector} s = sel.select()) {
    typecase(s) {
      when(SJSocket{?(D).?(D).!<R>} s1) { …; sel.register(s1); }
      when(SJSocket{?(D).!<R>} s2) { … }
} } }
```

ESP

$(\nu b)\ (b[\varepsilon] \mid \overline{b}(\overline{sel}).b(sel)$

$\overline{sel}!\langle s_1\rangle.\ldots.\overline{sel}!\langle s_n\rangle.\mathsf{def}\ \mathsf{Select}(x\overline{x})$

$= x?(y).\mathtt{if\ arrived}\ y\ \mathtt{then}$

$.\overline{x}!\langle s_1\rangle.\mathsf{Select}\langle x\overline{x}\rangle$

$\mathsf{Select}\langle x\overline{x}\rangle \quad \}$

$\mathtt{else}\ \overline{x}!\langle y\rangle.\mathsf{Select}\langle x\overline{x}\rangle$

$\mathtt{in}\ \mathsf{Select}\langle sel, \overline{sel}\rangle)$

Representing Selectors in ESP

ESP+

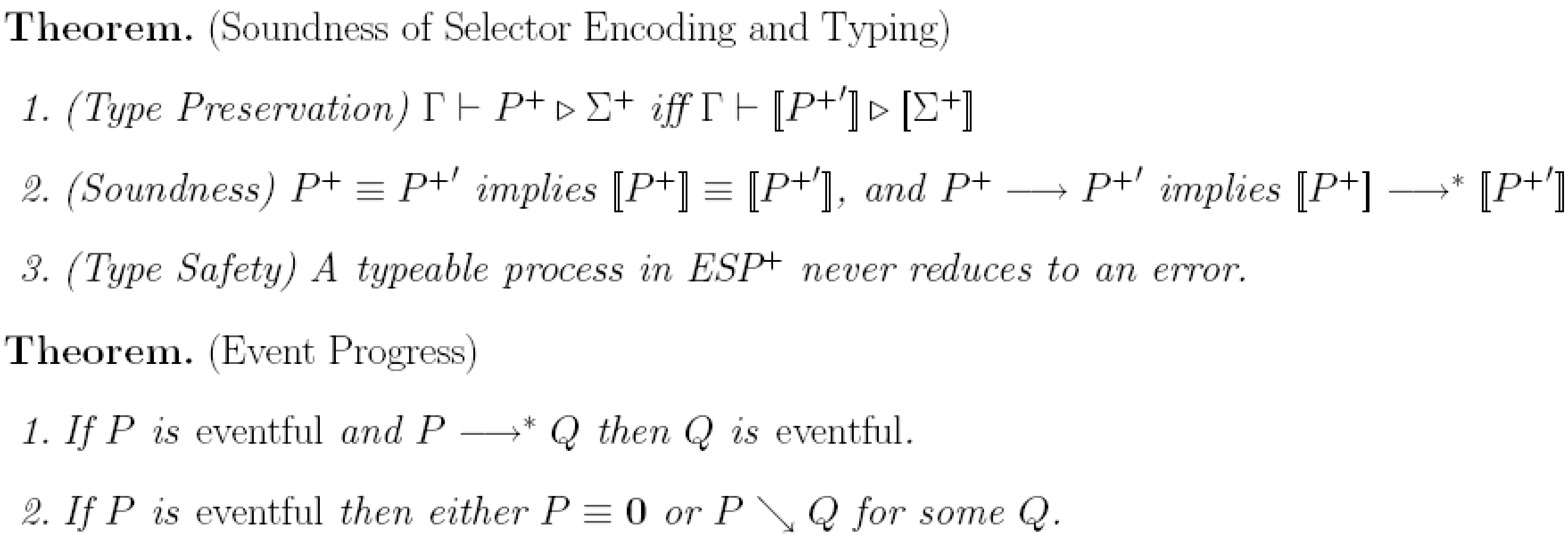
```
SJSelector{pSelector} sel = new SJSelector{pSelector}();
sel.register(source);
while(run) { // Main event loop.
  using(SJSocket{pSelector} s = sel.select()) {
    typecase(s) {
      when(SJSocket{?(D).?(D).!<R>} s1) { …; sel.register(s1); }
      when(SJSocket{?(D).!<R>} s2) { … }
} } }
```

ESP

$$
\begin{array}{l}
(\nu b)\,(b[\varepsilon] \mid \overline{b}(\overline{sel}).b(sel).\\
\quad \overline{sel}!\langle s_1\rangle.\ldots.\overline{sel}!\langle s_n\rangle.\mathsf{def}\ \mathsf{Select}(x\overline{x})\\
\qquad = x?(y).\mathtt{if\ arrived}\ x\ \mathtt{then}\\
\qquad\qquad \mathtt{typecase}\ y\ \{\ \ (x_1 : ?(U_1).?(U_1).!\langle U_2\rangle) : x_1?(y_1).\overline{x}!\langle s_1\rangle.\mathsf{Select}\langle x\overline{x}\rangle\\
\qquad\qquad\qquad\qquad (x_2 : ?(U_1).!\langle U_2\rangle) : \quad x_2?(y_2).x_2!\langle v\rangle.\mathsf{Select}\langle x\overline{x}\rangle \quad \}\\
\qquad\qquad \mathtt{else}\ \overline{x}!\langle y\rangle.\mathsf{Select}\langle x\overline{x}\rangle\\
\quad \mathsf{in}\ \mathsf{Select}\langle sel, \overline{sel}\rangle)
\end{array}
$$

Event-handling Safety and Progress

- Encode higher-level event programming constructs into ESP
  - Selectors and event loops
  - Join patterns for correlating multiple events



Theorem. (Soundness of Selector Encoding and Typing)

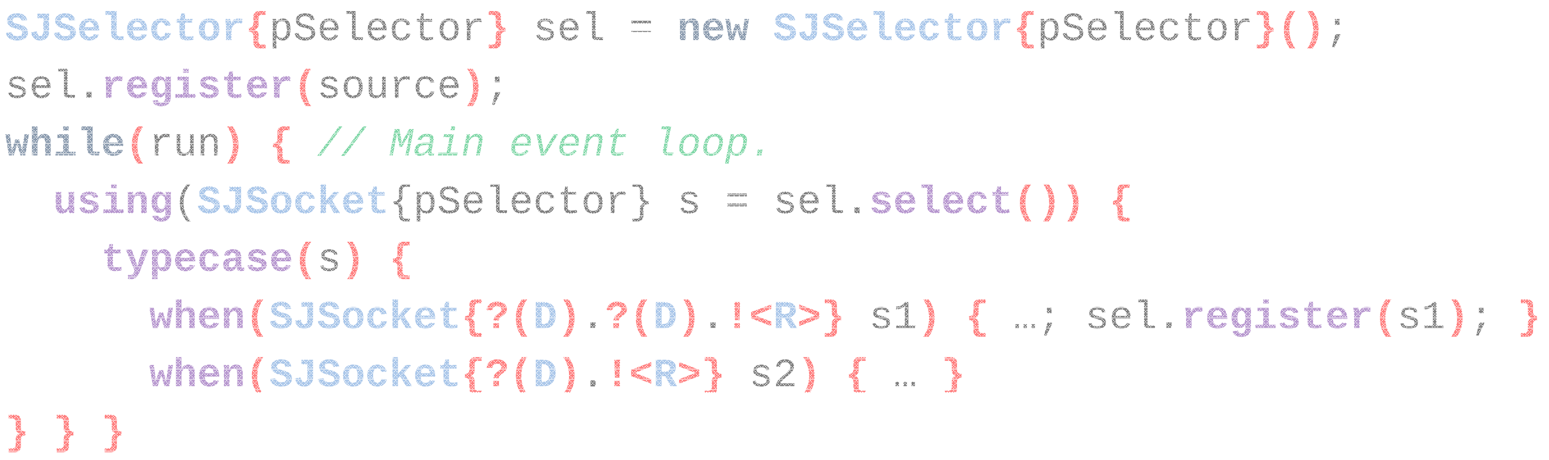
1. *(Type Preservation)* $\Gamma \vdash P^{+} \triangleright \Sigma^{+}$ *iff* $\Gamma \vdash [\![P^{+'}]\!] \triangleright [\![\Sigma^{+}]\!]$
2. *(Soundness)* $P^{+} \equiv P^{+'}$ *implies* $[\![P^{+}]\!] \equiv [\![P^{+'}]\!]$*, and* $P^{+} \longrightarrow P^{+'}$ *implies* $[\![P^{+}]\!] \longrightarrow^{*} [\![P^{+'}]\!]$
3. *(Type Safety) A typeable process in* ESP^{+} *never reduces to an error.*

Theorem. (Event Progress)

1. *If* P *is* eventful *and* $P \longrightarrow^{*} Q$ *then* Q *is* eventful.
2. *If* P *is* eventful *then either* $P \equiv \mathbf{0}$ *or* $P \searrow Q$ *for some* Q*.*

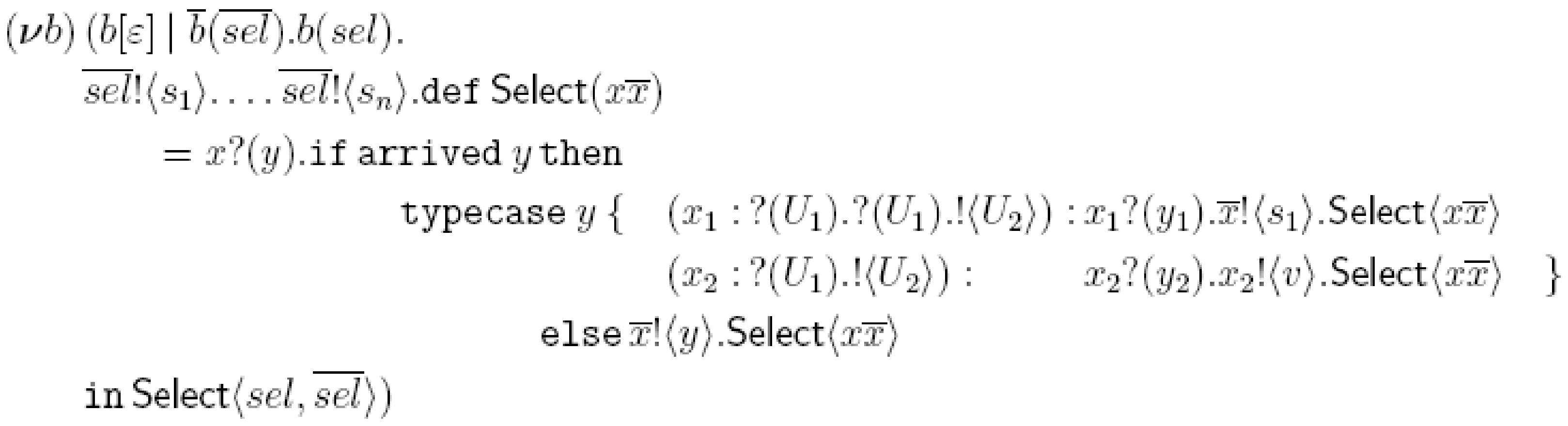
Representing Selectors in ESP

ESP+



```
SJSelector{pSelector} sel = new SJSelector{pSelector}();
sel.register(source);
while(run) { // Main event loop.
  using(SJSocket{pSelector} s = sel.select()) {
    typecase(s) {
      when(SJSocket{?(D).?(D).!<R>} s1) { …; sel.register(s1); }
      when(SJSocket{?(D).!<R>} s2) { … }
} } }
```

ESP



$$
\begin{array}{l}
(\nu b)\,(b[\varepsilon] \mid \overline{b}(\overline{sel}).b(sel).\\
\quad \overline{sel}!\langle s_1\rangle.\ldots.\overline{sel}!\langle s_n\rangle.\mathsf{def}\ \mathsf{Select}(x\overline{x})\\
\qquad = x?(y).\mathtt{if}\ \mathtt{arrived}\ y\ \mathtt{then}\\
\qquad\qquad \mathtt{typecase}\ y\ \{\ \ (x_1 : ?(U_1).?(U_1).!\langle U_2\rangle) : x_1?(y_1).\overline{x}!\langle s_1\rangle.\mathsf{Select}\langle x\overline{x}\rangle\\
\qquad\qquad\qquad\qquad (x_2 : ?(U_1).!\langle U_2\rangle) : \quad x_2?(y_2).x_2!\langle v\rangle.\mathsf{Select}\langle x\overline{x}\rangle\ \ \}\\
\qquad\qquad\quad \mathtt{else}\ \overline{x}!\langle y\rangle.\mathsf{Select}\langle x\overline{x}\rangle\\
\quad \mathsf{in}\ \mathsf{Select}\langle sel, \overline{sel}\rangle)
\end{array}
$$

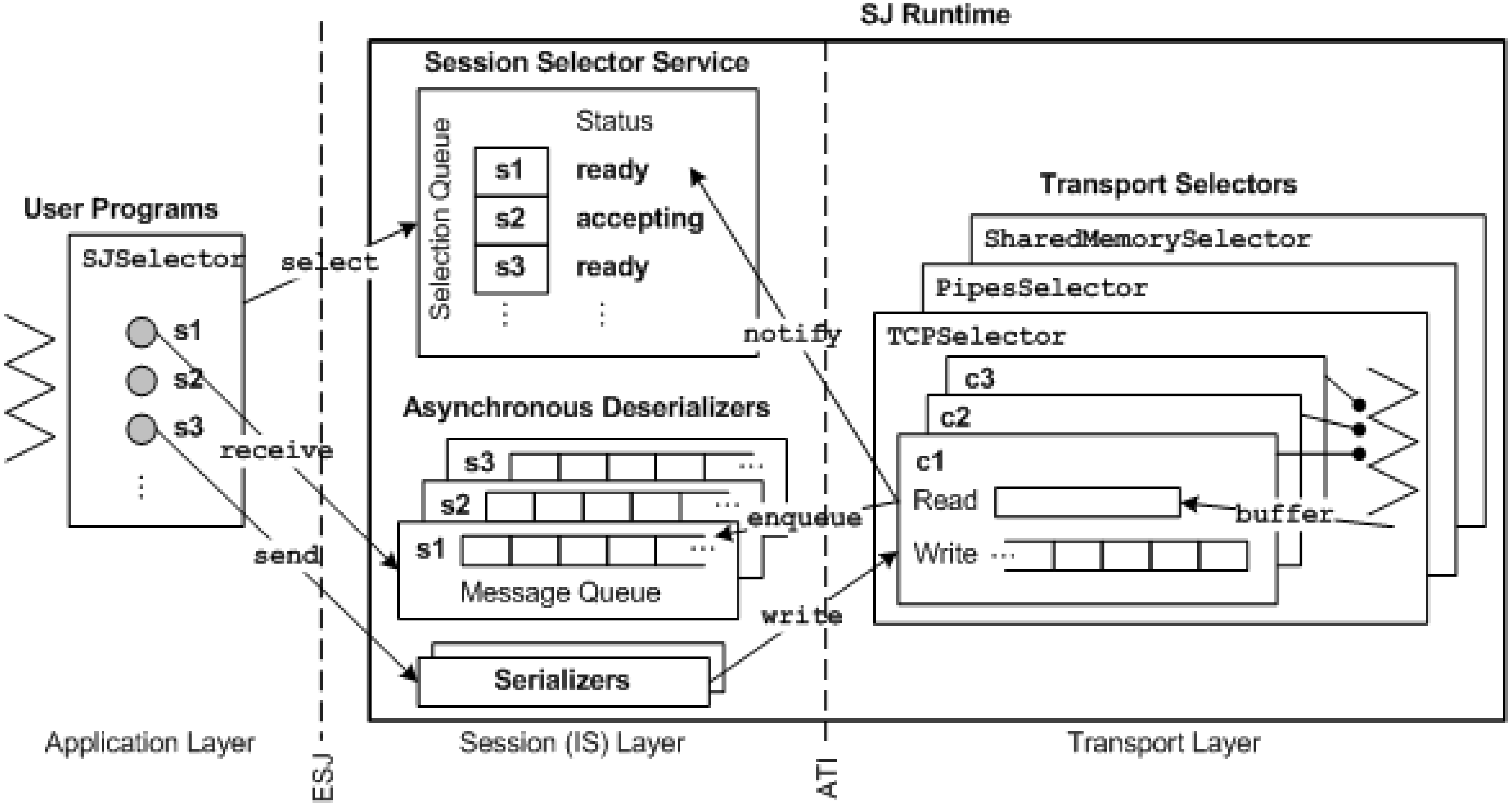
Type-safe Eventful Sessions in Java

```
begin
.!< Background, Contributions >
.!< Basic Example >
.!< Formalism and Properties >

.!< Implementation, Real-world Example: SMTP >

.!< Conclusion >
.?[
        ?( Question ).!< Answer >
]*
.end
```

SJSelector API Implementation



Session-typed SMTP Server

```
protocol pSmtpServer {
  !<Greeting>        220 smtp2.cc.ic.ac.uk SMTP Exim 4.69 …
  .?(Ehlo)           EHLO myname.doc.ic.ac.uk
  .!<EhloAck>        250-smtp2.cc.ic.ac.uk Hello myname…
  .@pBody
}
```

- Real-world example: branch types and recursion
- Session initiation events, branch events, …

Session-typed SMTP Server

```
protocol pSmtpServer {
  !<Greeting>         220 smtp2.cc.ic.ac.uk SMTP Exim 4.69 …
  .?(Ehlo)            EHLO myname.doc.ic.ac.uk
  .!<EhloAck>         250-smtp2.cc.ic.ac.uk Hello myname…
  .@pBody
}
```

Session-typed SMTP Server

```
protocol pBody {
  rec LOOP [
    ?{ // SMTP commands.
      MAIL: @pMail.#LOOP,
      RCPT: @pRcpt.#LOOP,
      DATA: @pData.#LOOP,
      …,
      QUIT: !<QuitAck>
    }
  ]
}
```

```
MAIL FROM:<rhu@doc.ic.ac.uk>
RCPT TO:<alice@doc.ic.ac.uk>
DATA … Dear Alice, …
```

Session-typed SMTP Server

```
protocol pSmtpServer {
  !<Greeting>
  .?(Ehlo)
  .!<EhloAck>
  .@pBody
}

protocol pMail {
  ?(Address)
  .!{ // Reply codes.
    RC250: !<AddrAck>,
    RC550: !<AddrError>,
    …
  }
}
```

```
...

protocol pBody {
  rec LOOP [
    ?{ // SMTP commands.
      MAIL: @pMail.#LOOP,
      RCPT: @pRcpt.#LOOP,
      DATA: @pData.#LOOP,
      …,
      QUIT: !<QuitAck>
    }
  ]
}
```

SMTP Events

```
protocol pSmtpServer {
  !<Greeting>
  .?(Ehlo)
  .!<Ehlo
  .@pBody
}

protocol
  ?(Addre
  .!{ //
    RC250
    RC550
    ...
  }
}
```

```
  ...

  protocol pBody {
```

```
protocol pSmtpEvents {
  sbegin.@pSmtpServer,          // Initiation event.
  ?(Ehlo).!<EhloAck>.@pBody,    // EHLO event.
  @pBody,                       // Command event.
  @pMail.@pBody,                // MAIL address event.
  @pRcpt.@pBody,                // RCPT address event.
  …
}
```

SMTP Event Loop (Outline)

```
void mainEventLoop(SJSelector{pSmtpEvents} sel) throws … {
  while(run) {
    using(SJChannel{@pSmtpEvents} c = sel.select()) {
      typecase(c) {
        …
        when(SJSocket{?(Ehlo).!<EhloAck>.@pBody} s1) { // EHLO.
          …
        }
        when(SJSocket{@pBody} s2) { // Main transaction loop.
          …
        }
        when(SJSocket{@pMail.@pBody} s3) { // MAIL address event.
          handleMail(s3);
          sel.register(s3);
        }
        …
} } } }
```

SMTP Event Loop (Outline)

```
void mainEventLoop(SJSelector{pSmtpEvents} sel) throws … {
  while(run) {
    using(SJChannel{@pSmtpEvents} c = sel.select()) {
      typecase(c) {
        …
        when(SJSocke                                                  o.
          …
        }
        when(SJSocke
          …
        }
        when(SJSocket{@pMail.@pBody} s3) { // MAIL address event.
          handleMail(s3);
          sel.register(s3);
        }
        …
} } } }
```

…groan… ☺

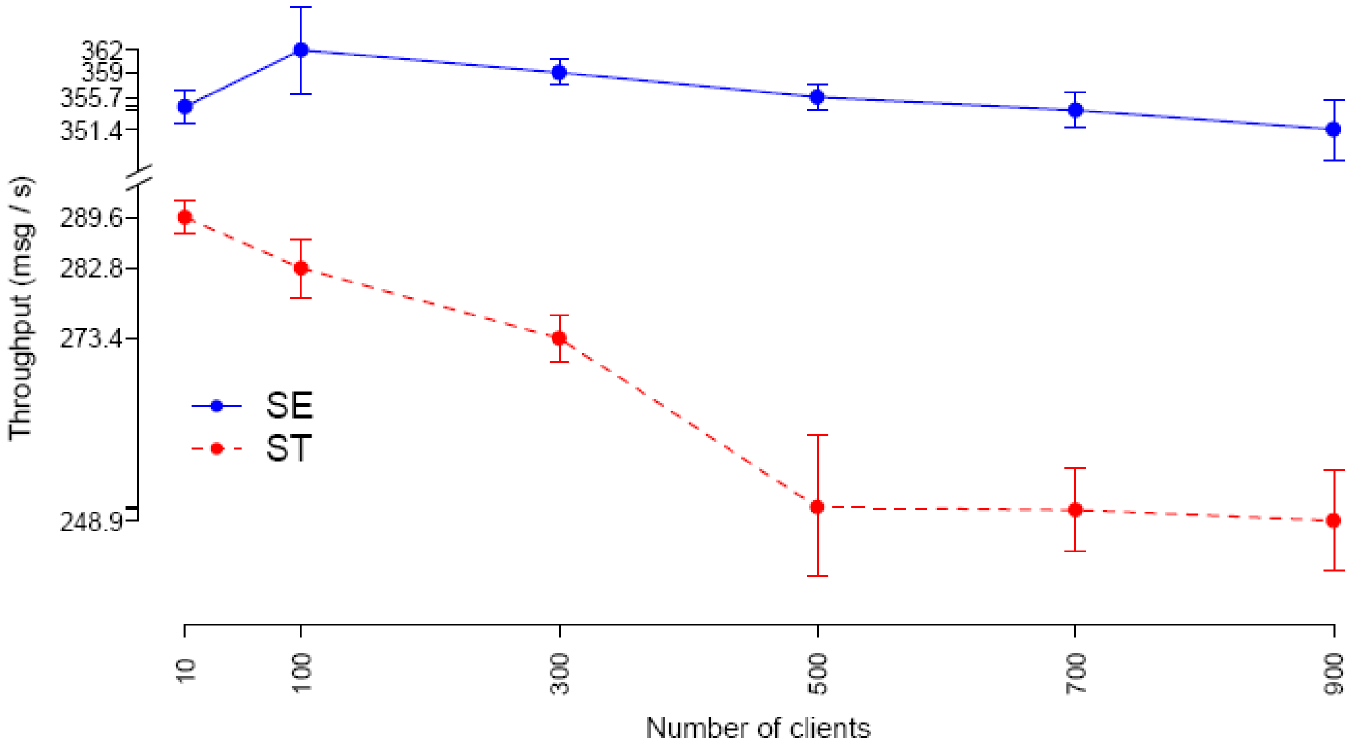
SMTP Event Loop (Outline)

```
void mainEventLoop(SJSelector{pSmtpEvents} sel) throws … {
  while(run) {
    using(SJChannel{@pSmtpEvents} c = sel.select()) {
      typecase(c) {
        …
        when(SJSocke                                                  O.
          …
        }
        when(SJSocke
          …
        }
        when(SJSocket{@pMail.@pBody} s3) { // MAIL address event.
          handleMail(s3);
          sel.register(s3);
        }
        …
} } } }
```

- Runtime session type monitoring
  + custom serialization components
  → maintain session type safety while being
  interoperable with non-SJ parties

Macro-benchmark (SMTP)

- Measure server throughput (#messages handled) under load from 10—900 concurrent clients



Conclusion

- *Session-typed event-driven programming*
  - Combined session types and event-driven programming
  - Formal model: event-handling safety and event progress

- Downloads (implementation, applications, benchmarks, long version):

  http://www.doc.ic.ac.uk/~rhu/sessionj.html

Type-safe Eventful Sessions in Java

```
begin
.!< Background, Contributions >
.!< Basic Example >
.!< Formalism and Properties >
.!< Implementation, Real-world Example: SMTP >
.!< Conclusion >

.?[
        ?( Question ).!< Answer >
]*
.end
```

Conclusion

- *Session-typed event-driven programming*
  - Combined session types and event-driven programming
  - Formal model: event-handling safety and event progress

- Downloads (implementation, applications, benchmarks, long version):

  http://www.doc.ic.ac.uk/~rhu/sessionj.html

Future Work

- Extensions for multiparty session types
  - Combine with “multiparty event handling” features, e.g. EventJava
- Session-typed selector thread pools
  - Exploit session types for thread locality and load balancing
- Adapt asynchronous session programming for session suspend/resume and process migration