# Eventful Sessions: Types, Programming and Bisimilarity

Raymond Hu, Dimitrios Kouzapas,
Olivier Pernet, Nobuko Yoshida

Kohei Honda

Imperial College London

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

# Outline

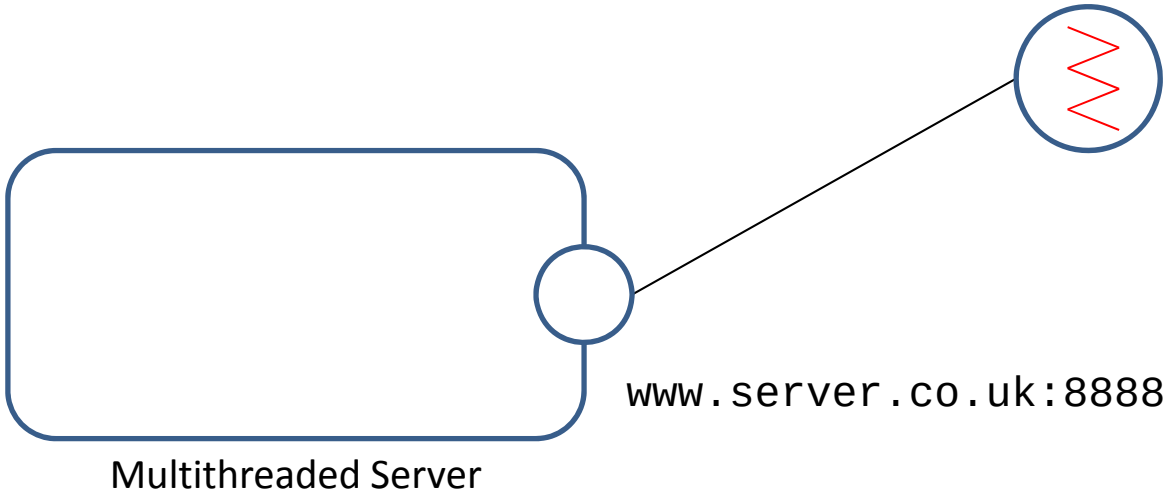
- Extend SJ (Session Java) for *session-typed event programming*.
  (ECOOP ‘10)

- Supporting formalisation of the key mechanisms in a minimal process model; *communication safety* for eventful session processes.

- Behavioural theory for asynchronous, eventful processes.
  (FMOODS ‘11)

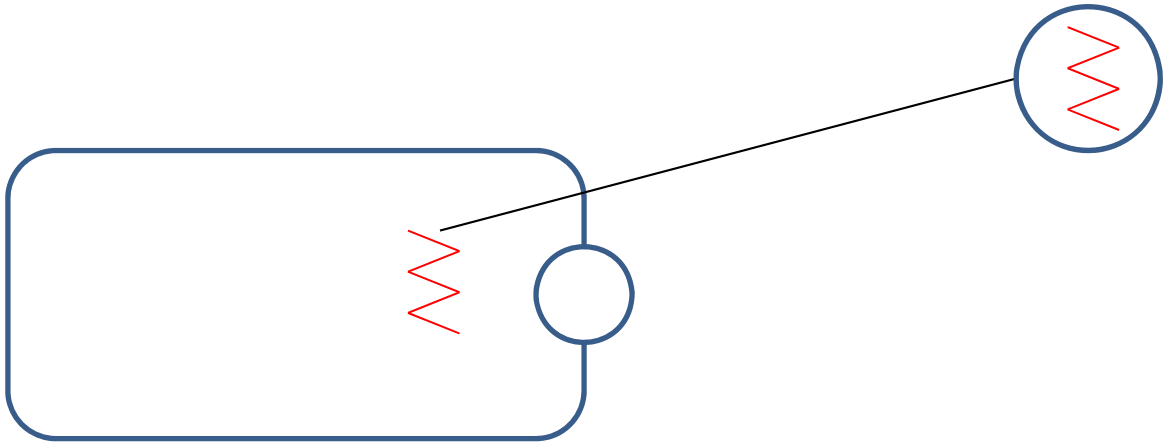
# Event-driven Concurrency

- … vs. multithreaded concurrency



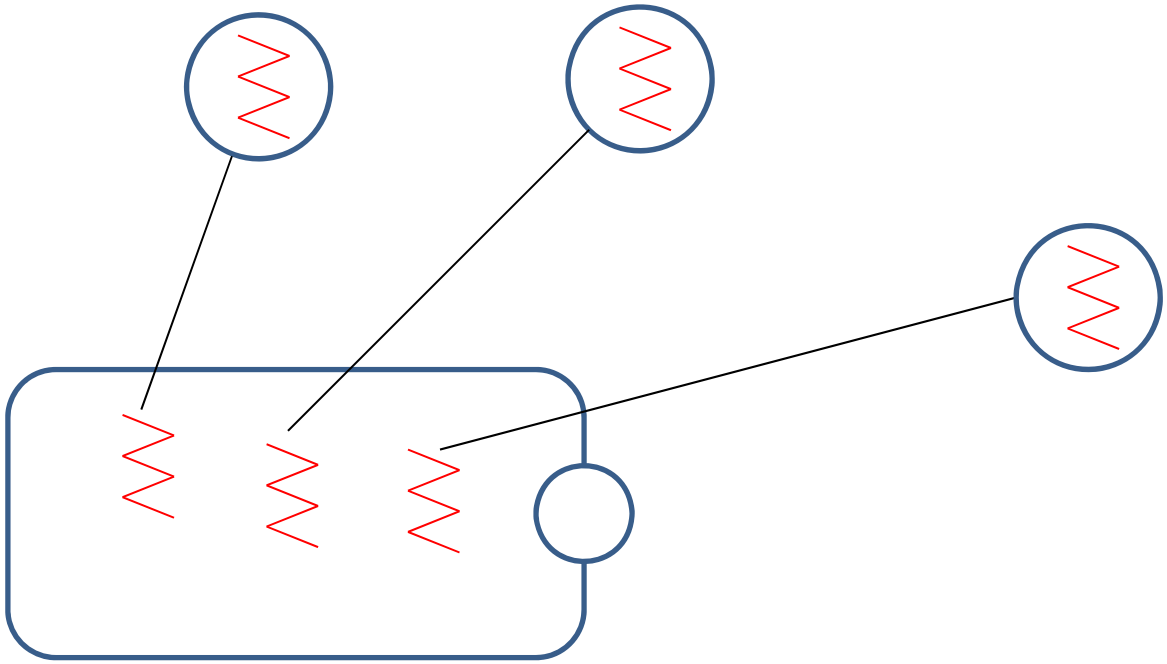
# Event-driven Concurrency

- … vs. multithreaded concurrency



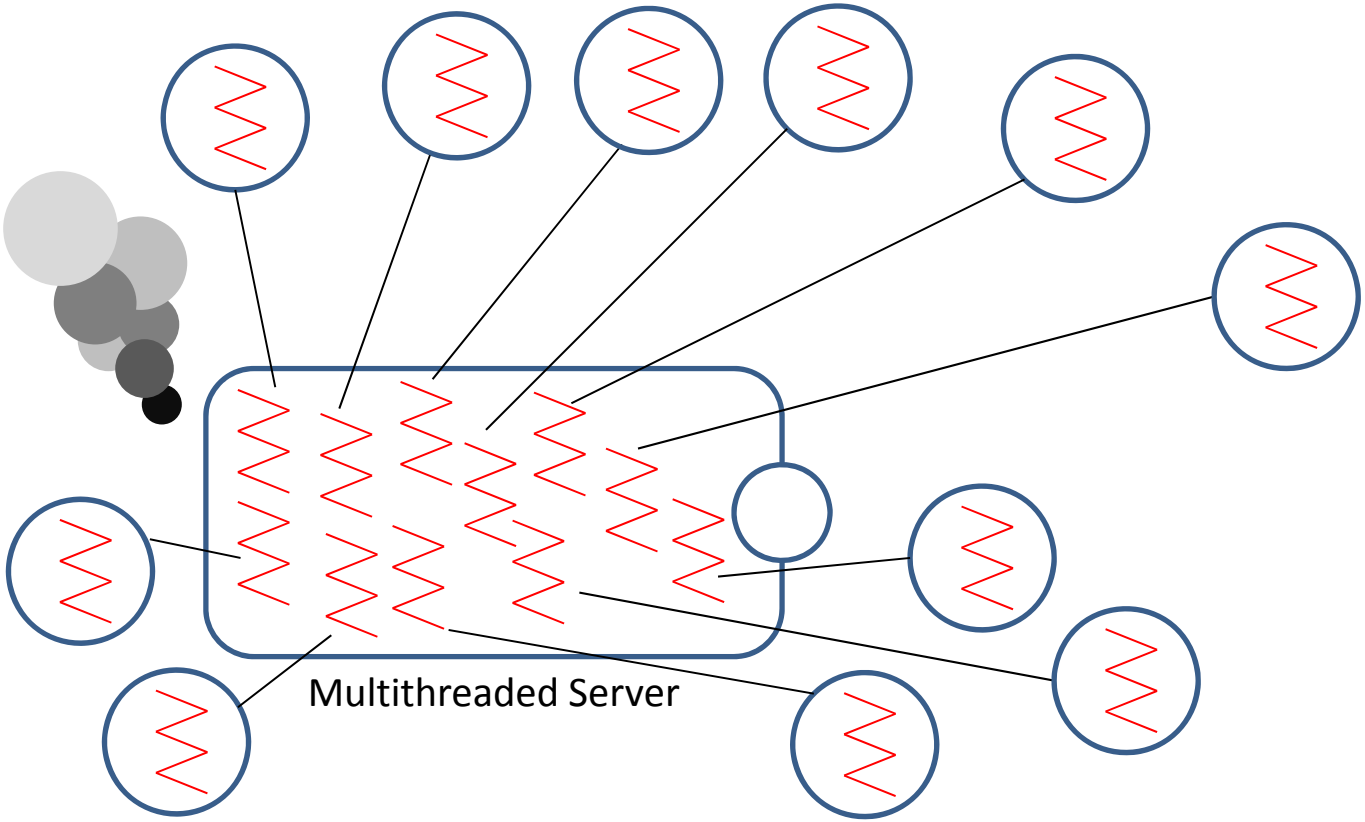
# Event-driven Concurrency

- … vs. multithreaded concurrency

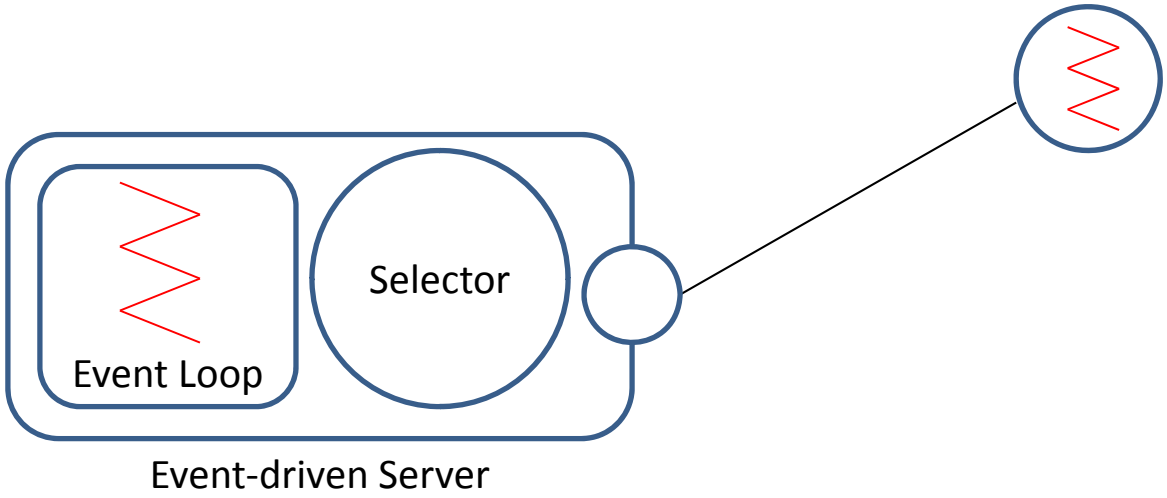


# Event-driven Concurrency

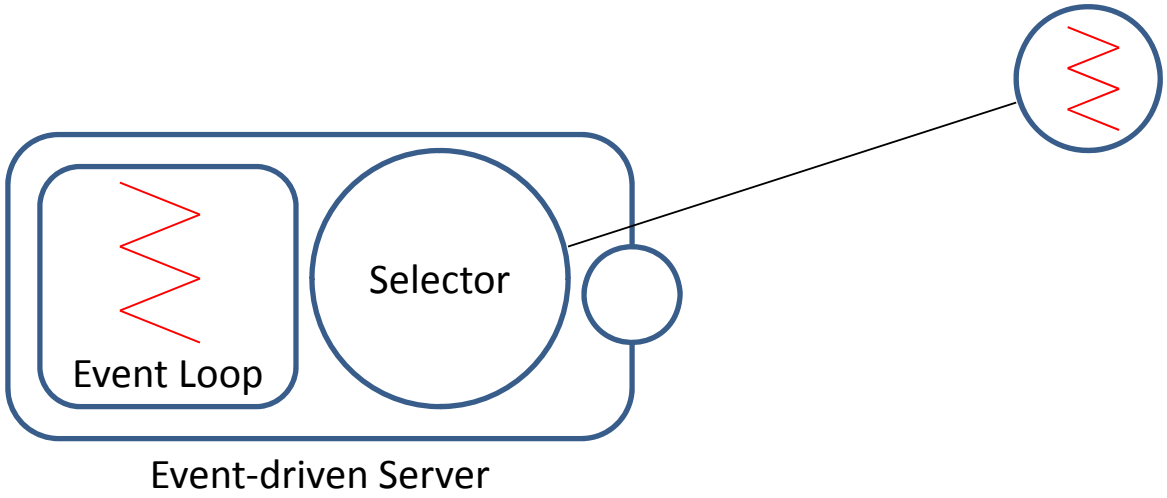
- … vs. multithreaded concurrency



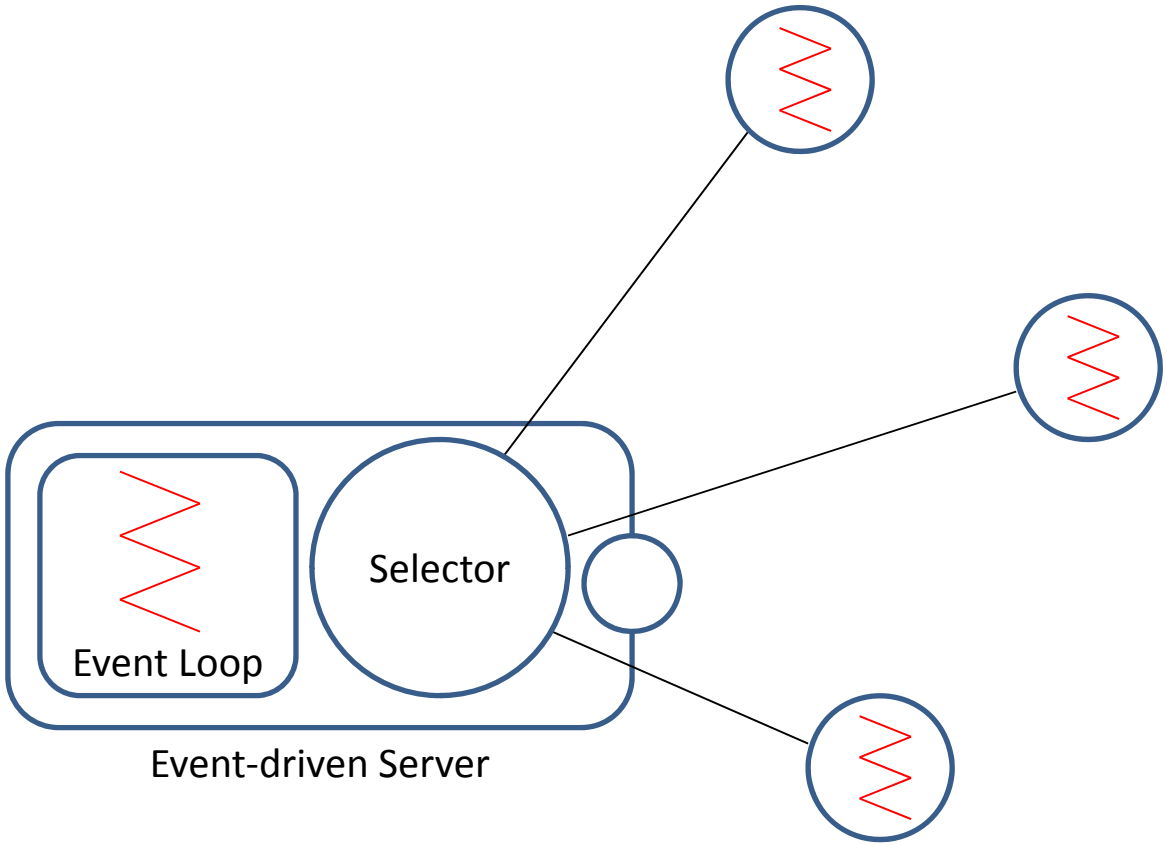
# Event-driven Concurrency



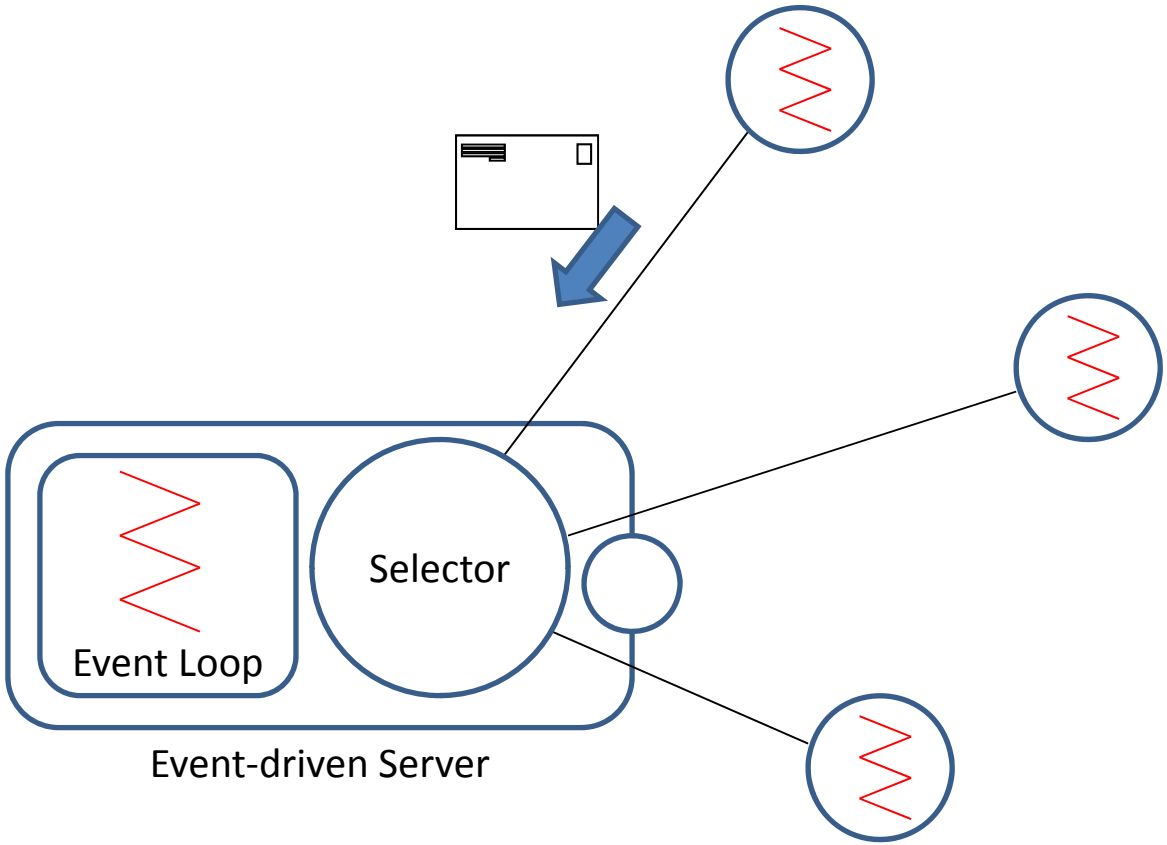
# Event-driven Concurrency



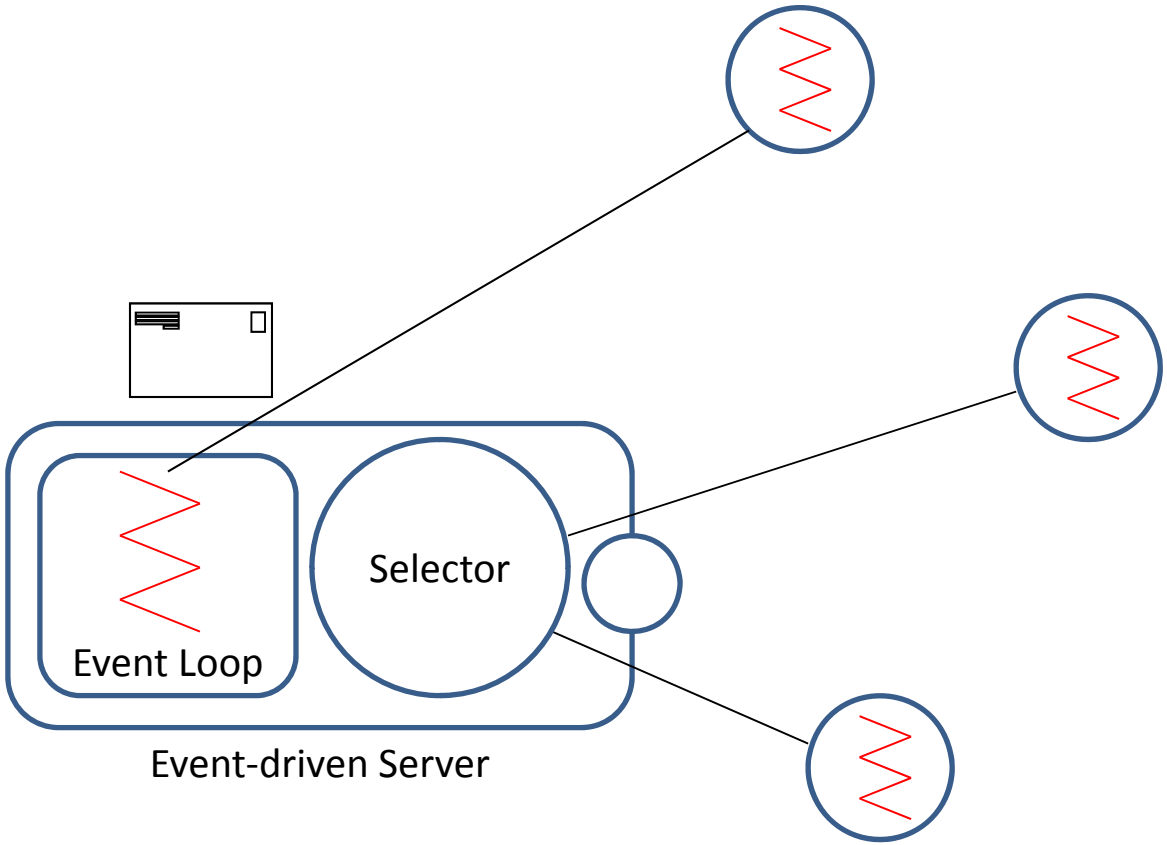
# Event-driven Concurrency



# Event-driven Concurrency

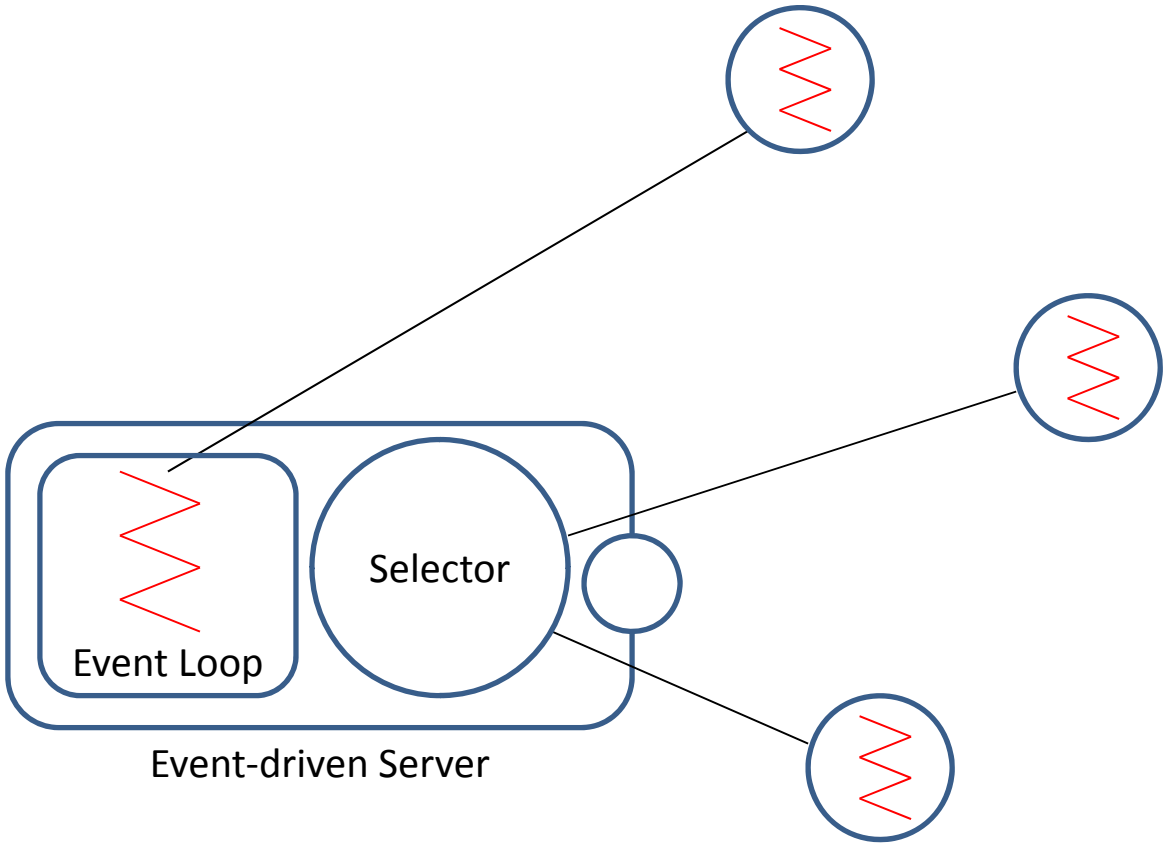


# Event-driven Concurrency



# Event-driven Concurrency

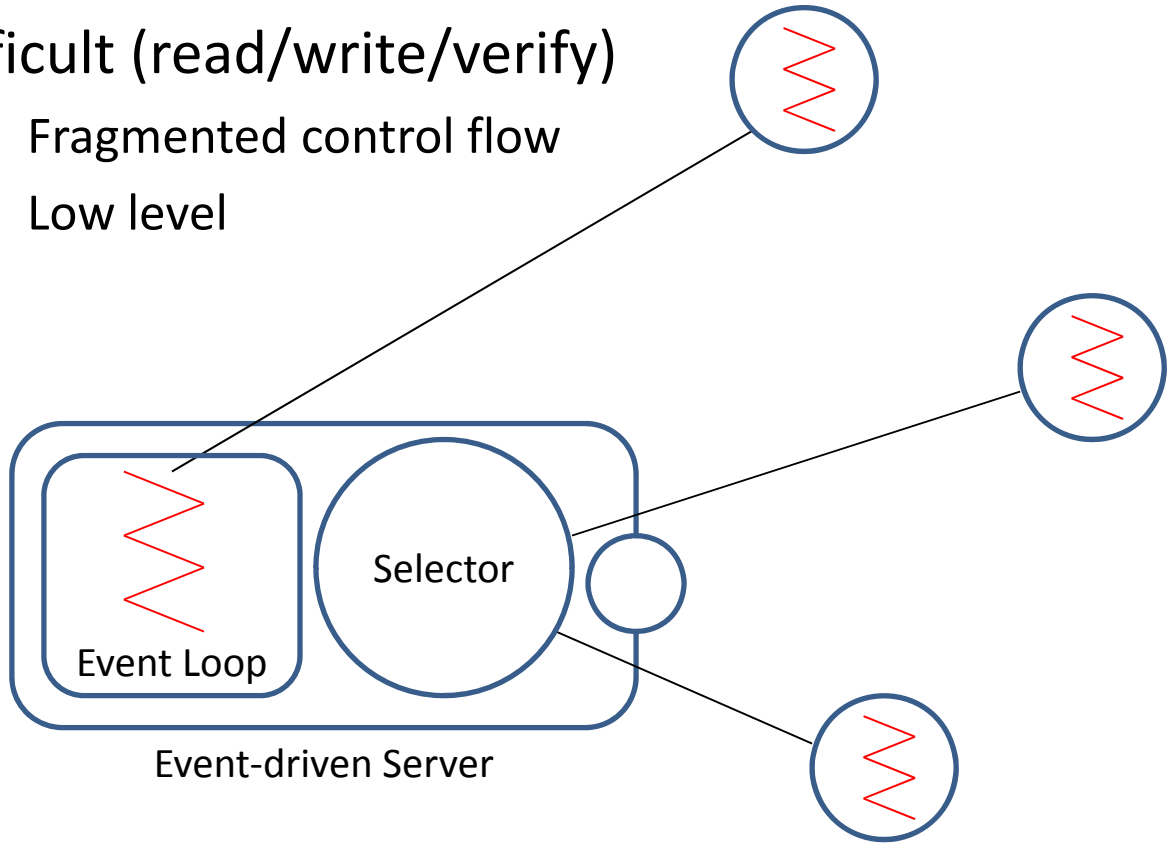
+ scalability



# Event-driven Concurrency

+ scalability

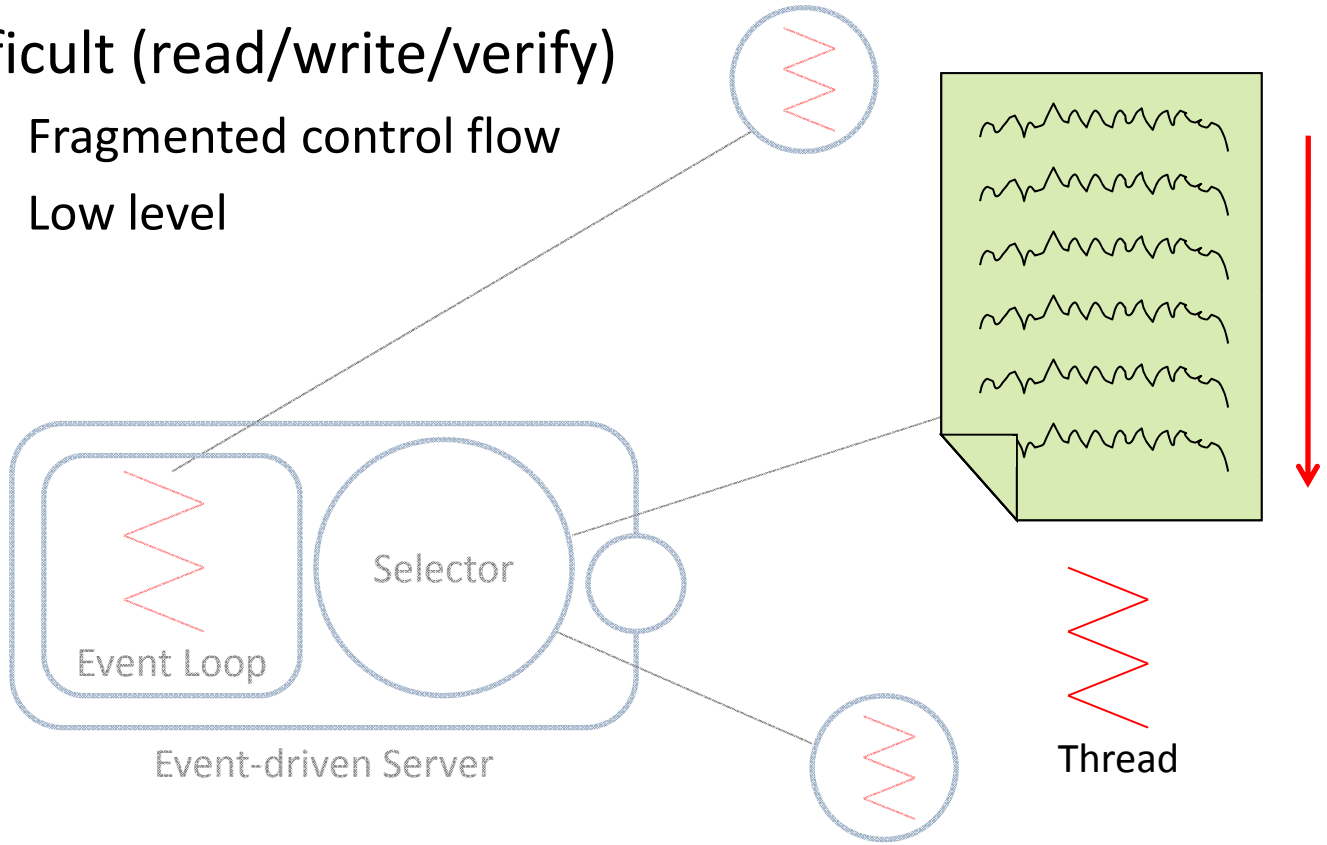
– difficult (read/write/verify)
  – Fragmented control flow
  – Low level



# Event-driven Concurrency

+ scalability

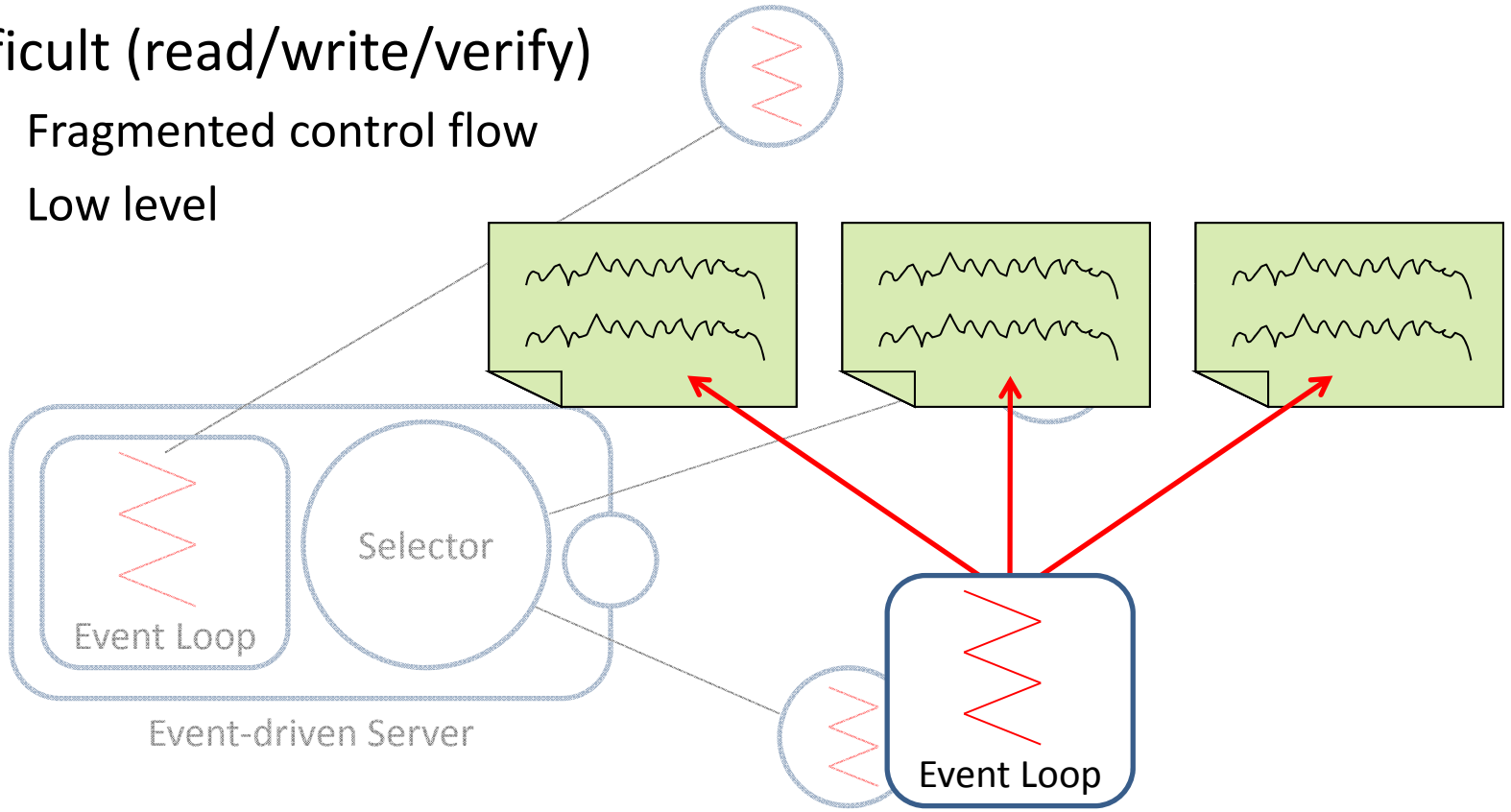
– difficult (read/write/verify)
  – Fragmented control flow
  – Low level



# 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 Programming

- Handle concurrent sessions of type:

!?(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 Programming

- Handle concurrent sessions of type:

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

# Session-typed Programming

- Handle concurrent sessions of type:

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

# Session-typed Programming

- Handle concurrent sessions of type:

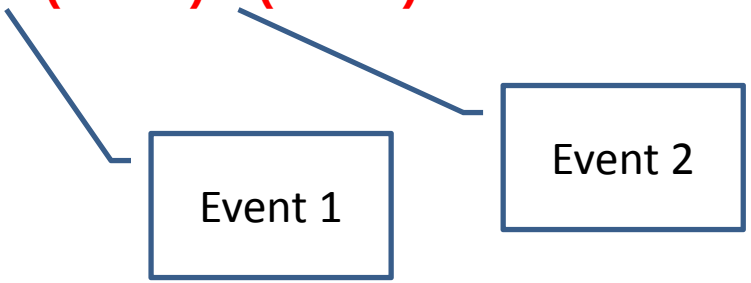
```
protocol pServer {  ?(Data).?(Data).!<Result>  }

...
SJSocket{pServer} s = ss.accept();
...
Data d1 = s.receive();      // ?(Data)
Data d2 = s.receive();      // ?(Data)
Result r = doSomeWork(d1, d2);
s.send(r);                  // !<Result>
```

# Session-typed Event Programming

- Handle concurrent sessions of type:

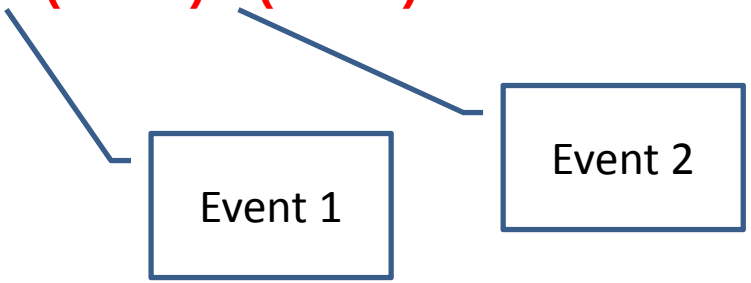
```
protocol pServer {  ?(Data).?(Data).!<Result>  }
```



# Session-typed Event Programming

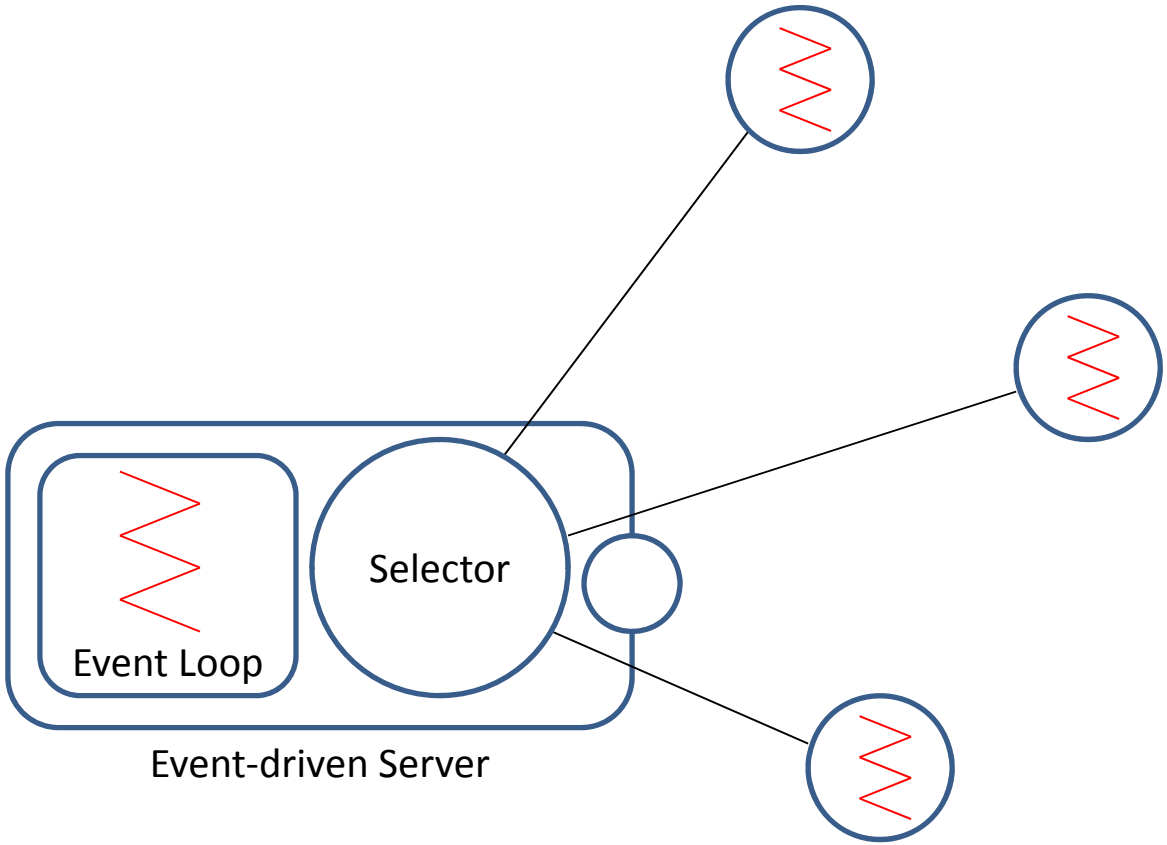
- Handle concurrent sessions of type:

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




```
// 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

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

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

```
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(...));
          }
} } } } }
```

Explicit specification of system events

Precise specification of each event

Correct matching of events to event handlers

# Session-typed Event Pr

```
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(...));
          }
} } } } }
```

Explicit specification of system events

Precise specification of each event

Correct matching of events to event handlers

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

# A Typing System for the Eventful Session $\pi$-Calculus

**Typing judgement for ESP**

$$\Gamma \vdash P \triangleright \Delta$$

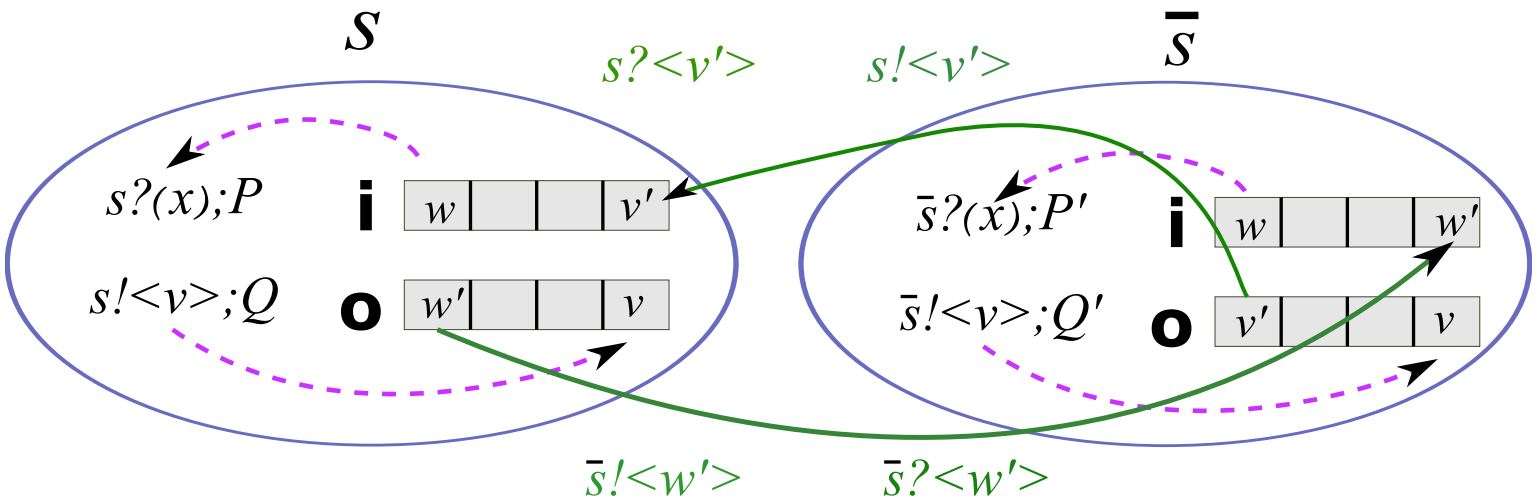
where

$$\begin{array}{lcl}
\Gamma & ::= & \emptyset \mid \Gamma \cdot v : S \mid \Gamma \cdot \mathbf{X} : \Delta \\
\Delta & ::= & \emptyset \mid \Delta \cdot k : \mathsf{T} \mid \Delta \cdot a \\
U & ::= & \mathtt{bool} \mid \mathtt{i}\langle S \rangle \mid \mathtt{o}\langle S \rangle \mid \mathbf{X} \mid \mu \mathbf{X}.U \qquad T ::= U \\
S & ::= & !\langle T \rangle; S \mid ?(T); S \mid \oplus \{l_i : S_i\}_{i \in I} \mid \& \{l_i : S_i\}_{i \in I} \\
 & \mid & \{S_i\}_{i \in I} \mid \mu \mathbf{X}.S \mid \mathbf{X} \mid \mathtt{end}
\end{array}$$

**Subject Reduction Theorem**

Let $\Gamma \vdash P \triangleright \Delta$ with $\mathtt{wc}(\Delta)$. Then if $P \twoheadrightarrow Q$ then $\Gamma \vdash Q \triangleright \Delta'$ with $\Delta \twoheadrightarrow \Delta'$ and $\mathtt{wc}(\Delta')$.

# Labeled Transition System




**Local** (the dashed arrows)

$$s!\langle v\rangle; Q \mid s[\mathtt{o} : \vec{h}] \xrightarrow{\tau} Q \mid s[\mathtt{o} : \vec{h} \cdot v]$$
$$s?(x); P \mid s[\mathtt{i} : w \cdot \vec{h}] \xrightarrow{\tau} P\{w/x\} \mid s[\mathtt{i} : \vec{h}]$$

**Remote** (the solid arrows)

$$s[\mathtt{i} : \vec{h}] \xrightarrow{s?\langle v\rangle} s[\mathtt{i} : \vec{h} \cdot v]$$
$$s[\mathtt{o} : v \cdot \vec{h}] \xrightarrow{s!\langle v\rangle} s[\mathtt{o} : \vec{h}]$$

# Typed Transition/Reduction - Localisation

- **Typed transition:** $\Gamma \vdash P \triangleright \Delta \xrightarrow{\ell} P' \triangleright \Delta'$ if
  1. $P \xrightarrow{\ell} P'$ (ensures asynchrony)
  2. $(\Gamma, \Delta) \xrightarrow{\ell} (\Gamma', \Delta')$ (ensures linearity)
- $P$ is **localised** with respect to $\Gamma, \Delta$ if
  1. For each $s : S \in \mathtt{dom}(\Delta)$, $s \in \Delta$
  2. if $\Gamma(a) = \mathtt{i}\langle S \rangle$, then $a \in \Delta$.
- A relation $\Gamma \vdash P \triangleright \Delta_1 \; \mathcal{R} \; \Gamma \vdash Q \triangleright \Delta_2$ is called **typed relation** if
  1. $P, Q$ localised
  2. $\Delta_1 \longrightarrow^* \Delta_2$ or $\Delta_2 \longrightarrow^* \Delta_1$.

# Typed Relations

- A typed relation $\mathcal{R}$ is called **typed reduction congruence** if whenever $P\mathcal{R}Q$ then:
  1. If $P \downarrow a$ then $Q \Downarrow a$
  2. If $\Gamma \vdash P \triangleright \Delta_1 \longrightarrow P \triangleright \Delta_1'$ then $\Gamma \vdash Q \triangleright \Delta_2 \twoheadrightarrow Q \triangleright \Delta_2'$ and $P\mathcal{R}Q$.
  3. $C[P]\mathcal{R}C[Q]$

  **Reduction congruency** ($\cong$) is the maximum (non-universal) reduction congruence.
- A typed relation $\mathcal{R}$ is a **weak asynchronous session bisimulation** if, whenever $\Gamma \vdash P_1 \triangleright \Delta_1 \mathcal{R} \Gamma \vdash P_2 \triangleright \Delta_2$, then
  1. $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} P_1' \triangleright \Delta_1'$ implies $\Gamma \vdash P_2 \triangleright \Delta_2 \stackrel{\hat{\ell}}{\Longrightarrow} P_2' \triangleright \Delta_2'$ such that $P_1'\mathcal{R}P_2'$.
  2. the symmetric case.

  **Bisimilarity** ($\approx$) is the maximum bisimulation.

# Theorems

- **Theorem** (Soundness and Completeness) $\approx = \cong$
- Asynchrony
  $s_1!\langle v_1\rangle; s_2!\langle v_2\rangle; \mathbf{0} \mid s_1[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon] \mid s_2[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon] \approx$
  $s_2!\langle v_2\rangle; s_1!\langle v_1\rangle; \mathbf{0} \mid s_1[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon] \mid s_2[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon]$.
- Linearity
  $s!\langle v_1\rangle; s!\langle v_2\rangle; \mathbf{0} \mid s_1[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon] \mid s_2[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon] \not\approx$
  $s!\langle v_2\rangle; s!\langle v_1\rangle; \mathbf{0} \mid s_1[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon] \mid s_2[\mathtt{i} : \epsilon, \mathtt{o} : \epsilon]$.
- Arrive Semantics
  Assume that $P \not\approx Q$, then
  $\mathtt{if\ arrive}\ s\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid s[\mathtt{i} : \epsilon] \mid \overline{s}[\mathtt{o} : v] \not\approx$
  $\mathtt{if\ arrive}\ s\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid s[\mathtt{i} : v] \mid \overline{s}[\mathtt{o} : \epsilon]$.
- Clear distinctions between behavioural semantics for the synchronous and the asynchronous $\pi$-calculi.

# Thread Elimination

- Lauer and Needham in 1979 observed that there is a **duality** between threaded shared memory systems and event based message passing systems.
- A simple server has the form $*\,a(x).P \mid a[\epsilon]$ where $P$ is confluent and handles each client in a sequential way.
- Define a transformation $LN[\![*\,a(x).P \mid a[\epsilon]]\!]$ that handles clients on message reception events.
  - Events are identified as the blocking session actions.
  - Use a selector construct to select next client/session to handle.

# Thread Elimination: Mapping

$$LN[\![* a(x).P]\!] \stackrel{\text{def}}{=} (\nu\ o, q, \vec{c})(\mathsf{Loop}\langle o, q\rangle \mid \overline{o} \mid q\langle a, c_0\rangle \mid \mathsf{CodeBlocks}\langle a, o, q, \vec{c}\rangle)$$

$$\begin{aligned}\mathsf{Loop}\langle o, q\rangle \stackrel{\text{def}}{=}\ & * o.\texttt{select}\ x\ \texttt{from}\ q\ \texttt{in}\ \texttt{typecase}\ x\ \texttt{of}\ \{\\ & (x : S,\ z : \mathsf{env}) : \texttt{new env}\ y\ \texttt{in}\ \overline{z}\langle y\rangle,\\ & (x : S_1,\ y : \mathsf{env},\ z : S_1, \mathsf{env}) :\ \overline{z}\langle x, y\rangle,\ \ldots\\ & (x : S_{n-m},\ y : \mathsf{env},\ z : S_{n-m}, \mathsf{env}) :\ \overline{z}\langle x, y\rangle\}\end{aligned}$$

$$\mathsf{CodeBlocks}\langle a, o, q, \vec{c}\rangle \stackrel{\text{def}}{=} \mathcal{B}[\![a(x).P]\!] \mid \prod\nolimits_{1\le i\le n} \mathcal{B}[\![P_i]\!]$$

$$\begin{aligned}
\mathcal{B}[\![* a(x).P]\!] &\stackrel{\text{def}}{=} * c_0.ya(w).\texttt{update}\ (\mathtt{y}, \mathtt{w}, \mathtt{w}')\ \texttt{in register}\ a, c_0\ \texttt{to}\ q\ \texttt{in}\ [\![P, y]\!]\\
\mathcal{B}[\![x^{(i)}?(z : T); Q]\!] &\stackrel{\text{def}}{=} * c_i(x', y).x'?(z'); \texttt{update}\ (\mathtt{y}, \mathtt{z}, \mathtt{z}')\ \texttt{in update}\ (\mathtt{y}, \mathtt{x}, \mathtt{x}')\ \texttt{in}\ [\![Q, y]\!]\\
\mathcal{B}[\![x^{(i)}\&\{l_j : Q_j\}_j]\!] &\stackrel{\text{def}}{=} * c_i(x', y).x'\&\{l_j :\ \texttt{update}\ (\mathtt{y}, \mathtt{x}, \mathtt{x}')\ \texttt{in}\ [\![Q_j, y]\!]\}_j\\
[\![x!\langle e\rangle; Q, y]\!] &\stackrel{\text{def}}{=} \texttt{let}\ x' = [\![x]\!]_y\ \texttt{in}\ x'!\langle [\![e]\!]_y\rangle; \texttt{update}\ (\mathtt{y}, \mathtt{x}, \mathtt{x}')\ \texttt{in}\ [\![Q, y]\!]\\
[\![x!\langle k\rangle; Q, y]\!] &\stackrel{\text{def}}{=} \texttt{let}\ x' = [\![x]\!]_y\ \texttt{in let}\ k' = [\![k]\!]_y\ \texttt{in}\ x'!\langle k'\rangle; \texttt{update}\ (\mathtt{y}, \mathtt{xk}, \mathtt{x'k'})\ \texttt{in}\ [\![Q, y]\!]\\
[\![x \oplus l_j; Q, y]\!] &\stackrel{\text{def}}{=} \texttt{let}\ z = [\![x]\!]_y\ \texttt{in}\ z \oplus l_j; [\![Q, y]\!]\\
[\![\overline{b}(z : S); Q, y]\!] &\stackrel{\text{def}}{=} \overline{b}(z' : S); \texttt{update}\ (\mathtt{y}, \mathtt{z}, \mathtt{z}')\ \texttt{in}\ [\![Q, y]\!]\\
[\![Q, y]\!] &\stackrel{\text{def}}{=} \texttt{let}\ x' = [\![x]\!]_y\ \texttt{in register}\ x', c_i, yq\overline{o}\ \ (Q \text{ is blocking at } x^{(i)})\\
[\![\mathbf{0}, y]\!] &\stackrel{\text{def}}{=} \overline{o}
\end{aligned}$$

# Thread Elimination

**Theorem:** $* a(x).P \mid a[\epsilon] \approx LN[\![* a(x).P \mid a[\epsilon]]\!]$

For proofs, we use *permutation*, *confluence* and *determinacy* [Philippou and Walker'97] guaranteed by session types.

- $* a(x).P \mid a[\epsilon]$ is confluent.
- $LN[\![* a(x).P \mid a[\epsilon]]\!]$ processes events/clients in a sequential way.

# Conclusions

- Eventful Session Typed $\pi$-Calculus.
- Session typed behavioural semantics.
- Bisimulation is sound and complete with respect to reduction-closed congruency.
- Permutations/confluence/determinacy properties enforced by session types.
- Use the properties of bisimulations to prove a semantic equivalence for a thread elimination transform [Lauer and Needham'79].
- D. Kouzapas, N. Yoshida and K. Honda. On Asynchronous Session Semantics. To appear in FMOODS '11.
  Full version with benchmarks in Eventful SJ:
  www.doc.ic.ac.uk/~dk208/semantics.html