# Moscova

Jean-Jacques Lévy

INRIA Paris–Rocquencourt

March 23, 2011

# Research team

# Stats

- Staff 2008-2011

Jean-Jacques Lévy, INRIA
Karthikeyan Bhargavan, INRIA
James Leifer, INRIA
Luc Maranget, INRIA
Francesco Zappa Nardelli, INRIA
Ricardo Corin, INRIA $\longrightarrow$ Cordoba
Gilles Peskine, INRIA $\longrightarrow$ Trusted logic
Pierre-Malo Deniélou, INRIA $\longrightarrow$ Imperial College
Jade Alglave, INRIA $\longrightarrow$ Oxford
Nataliya Guts, MSR-INRIA $\longrightarrow$ Maryland
Jérémy Planul, MSR-INRIA

- INRIA Rocquencourt $\longleftrightarrow$ MSR-INRIA Saclay (Cédric Fournet, MSRC)

- Moscova history:
  - Para (1988, Head: Lévy), Moscova (2000, Head: Gonthier $\longrightarrow$ MSRC)
  - 18 PhDs
  - 75% Coq proof of the 4-color thm; debugging of 3 modules of Ariane-501 PV; spinoff of Polyspace [Alain Deutsch]; etc.
  - Polytechnique (Lévy, prof 1992-2006) $\longrightarrow$ MSR-INRIA Joint Centre (Head: Lévy)

# Research themes

# Research themes

- programming languages
  [safe marshalling, ott, like types]
- concurrency
  [jocaml, separation logic/c-minor/concurrency, weak memory models]
- security compilers and verifiers
  [secure sessions, audits, tls, information flow]

# Research results

# Weak memory models

- memory models of multi-core processors
- give formal description of WMMs
- operational semantics of WMMs
- certified (back-end) compiler for some WMMs
- prove correctness of compiler optimisations in WMMs

# Intel whitepaper (1/3)

MP (message passing)

### 2.1 Loads are not reordered with other loads and stores are not reordered with other stores

Intel 64 memory ordering ensures that loads are seen in program order, and that stores are seen in program order.

| Processor 0 | Processor 1 |
|---|---|
| mov [ _x], 1 // M1<br>mov [ _y], 1 // M2 | mov r1,[_y] // M3<br>mov r2, [_x] // M4 |
| Initially x == y == 0<br>r1 == 1 and r2 == 0 is not allowed | |

Table 2.1: Stores are not reordered with other stores

[demo]

# Intel whitepaper (2/3)

SB (store buffers)

## 2.3 Loads may be reordered with older stores to different locations

Intel 64 memory ordering allows load instructions to be reordered with prior stores to a different location. However, loads are not reordered with prior stores to the same location.

The first example in this section illustrates the case in which a load may be reordered with an older store – i.e. if the store and load are to different non-overlapping locations.

| Processor 0 | Processor 1 |
|---|---|
| mov [ _x], 1 // M1<br>mov r1, [ _y] // M2 | mov [ _y], 1 // M3<br>mov r2, [_x] // M4 |
| Initially x == y == 0<br>r1 == 0 and r2 == 0 is allowed | |

Table 2.3.a: Loads may be reordered with older stores

[demo]

# Intel whitepaper (3/3)  WRC (write-to-read causality)

## 2.5 Stores are transitively visible

Intel 64 memory ordering ensures transitive visibility of stores – i.e. stores that are causally related appear to execute in an order consistent with the causal relation.

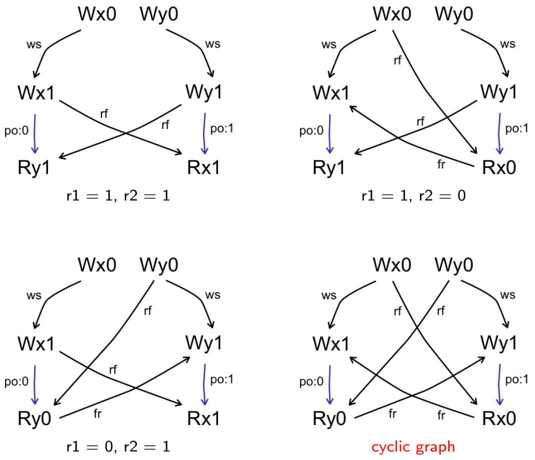
| Processor 0 | Processor 1 | Processor 2 |
|---|---|---|
| mov [ _x], 1 // M1 | mov r1, [ _x] // M2<br>mov [ _y], 1 // M3 | mov r2, [ _y] // M4<br>mov r3, [ _x] // M5 |
| Initially x == y == 0<br>r1 == 1, r2 == 1, r3 == 0 is not allowed | | |

Table 2.5: Stores are transitively visible

[demo]

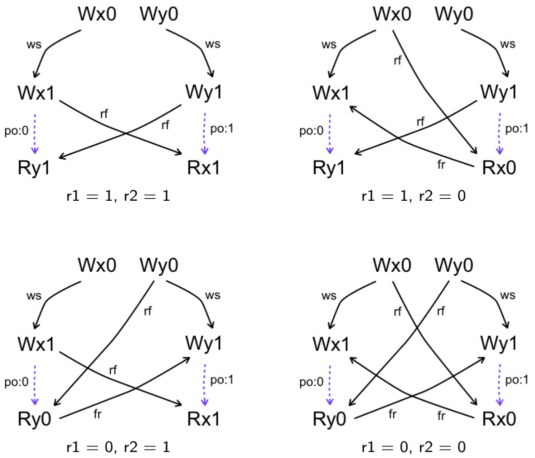
- In SC, program order is strictly respected.

| P0 | P1 |
|---|---|
| mov [ x], 1 | mov [ y], 1 |
| mov r1, [ y] | mov r2, [ x] |



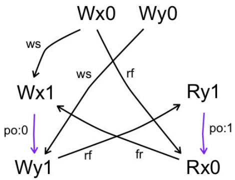
- In TSO, *W* followed by *R* can be relaxed within program order

| P0 | P1 |
|---|---|
| mov [ x], 1 | mov [ y], 1 |
| mov r1, [ y] | mov r2, [ x] |



- In TSO, *W* followed by *R* relaxed

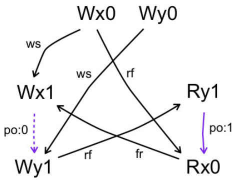
| P0 | P1 |
|---|---|
| mov [ x], 1 | mov r1, [ y] |
| mov [ y], 1 | mov r2, [ x] |




r1 = 1, r2 = 0 Cyclic graph

- In PSO, *W* followed by *W* to distinct location relaxed

| P0 | P1 |
|---|---|
| mov [ x], 1 | mov r1, [ y] |
| mov [ y], 1 | mov r2, [ x] |




r1 = 1, r2 = 0

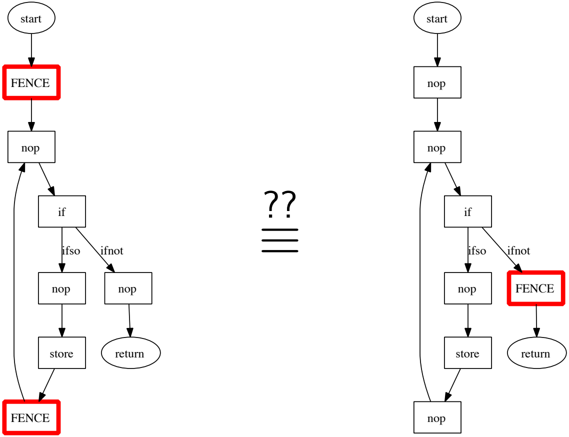
# Weak memory models

- axiomatic + operational models for Intel [~Cambridge] / Power [~INRIA]
- formalisation in HOL/Coq
- tests on real processor behaviour
  http://www.cl.cam.ac.uk/~pes20/ppc-supplemental/ppc003.html
- formal proof of simple concurrent code (eg. Linux spinlocks)
- operational reasoning: data-race freedom, separation logic
- certified compiler for concurrent languages
  http://www.cl.cam.ac.uk/~pes20/CompCertTSO

[Zappa Nardelli, Maranget, Alglave, Braibant, Sewell et al]
[POPL 09, CACM 10; DAMP 09, CAV 10, PLDI 11; TACAS 11; POPL 11]

# Weak memory models

- Proving correctness of optimisations




Fences elimination with TSO $\simeq$ 3 kloCoq

# Secure sessions

- passing authenticated (signed) values between 2 *run-times*
- design of a mini F# + primitives for authentication
  + global contract with sessions types
- compiling scheme into a low-level language ($\simeq$ pi-calculus)
  to describe authentication protocols
- formal proof of its correctness, with security property induced by
  strong typing of F# + usage of authentication primitives
- extension to other security properties, sessions V2
  (privacy of message values, integrity, dynamic number of principals, etc)

  [Corin, Deniélou, Leifer, Fournet, Bhargavan]

  [JCS 08, TGC 07, CSF 09, Deniélou phd 11]

# Secure sessions

- passing authenticated (signed) values between 2 *run-times*
- design of a mini F# + primitives for authentication
  + global contract with sessions types
- compiling scheme into a low-level language ($\simeq$ pi-calculus)
  to describe authentication protocols
- formal proof of its correctness, with security property induced by strong typing of F# + usage of authentication primitives
- extension to other security properties, sessions V2
  (privacy of message values, integrity, dynamic number of principals, etc)

  [Corin, Deniélou, Leifer, Fournet, Bhargavan]

  [JCS 08, TGC 07, CSF 09, Deniélou phd 11]

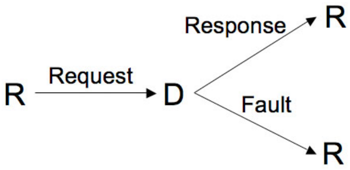
# Secure sessions

- passing authenticated (signed) values between 2 *run-times*
- design of a mini F# + primitives for authentication
  + global contract with sessions types
- compiling scheme into a low-level language ($\simeq$ pi-calculus)
  to describe authentication protocols
- formal proof of its correctness, with security property induced by strong typing of F# + usage of authentication primitives
- extension to other security properties, sessions V2
  (privacy of message values, integrity, dynamic number of principals, etc)

  [Corin, Deniélou, Leifer, Fournet, Bhargavan]

  [JCS 08, TGC 07, CSF 09, Deniélou phd 11]

# Simple exchange



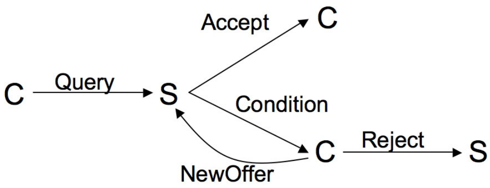

```
session S =
  role requester : int =
   !Request:string ;
   ?(Response:int + Fault:unit)

  role directory : string =
   ?Request:string;
   !(Response:int + Fault:unit)
```

Session declaration

```
let lookup name =
  S.requester ["client";"server"]
     (Request
         (name,
          {hResponse = (fun _ q → q) ;
           hFault = (fun _ x → failwith "Failed")
          }))
in lookup "Ricardo"
```

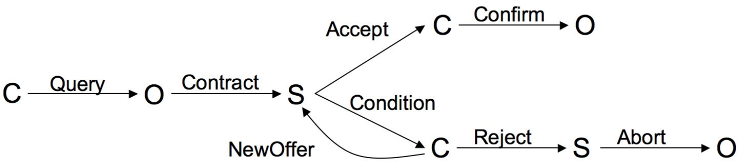
User code

# Two-party negotiation




```
session S2 =
  role customer : string =
   !Query:int;
     mu start.?(Accept:unit +
              Condition:unit;!(NewOffer:int;start + Reject:unit))

   role store : string=
   ?Query:int;
     mu start.!(Accept:unit +
              Condition:unit;?(NewOffer:int;start + Reject:unit))
```

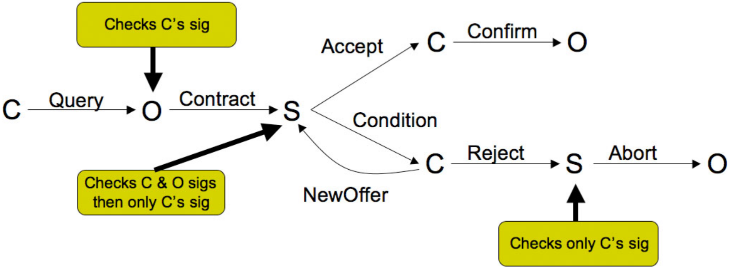
# Three-party session




```
session S3 =
  role customer :string =
   !Query:int;
    mu start.?(Accept:unit;!Confirm:unit +
              Condition:unit; !(Newoffer:int;start + Reject:unit;))

  role store :string=
   ?Contract:int;
    mu start.!(Accept:unit +
              Condition:unit; ?(Newoffer:int;start + Reject:unit;!Abort:unit))

  role officer :string=
   ?Query:int;!Contract:int;?(Confirm:unit + Abort:unit)
```

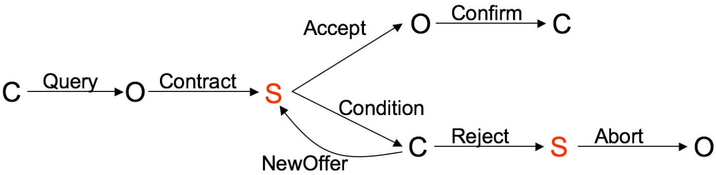
# Visibility

- Minimal sequence of signatures that guarantee session compliance.
- Example:

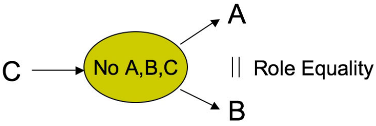


# No blind fork

- Some forks in protocols represent a security threat.




- Property



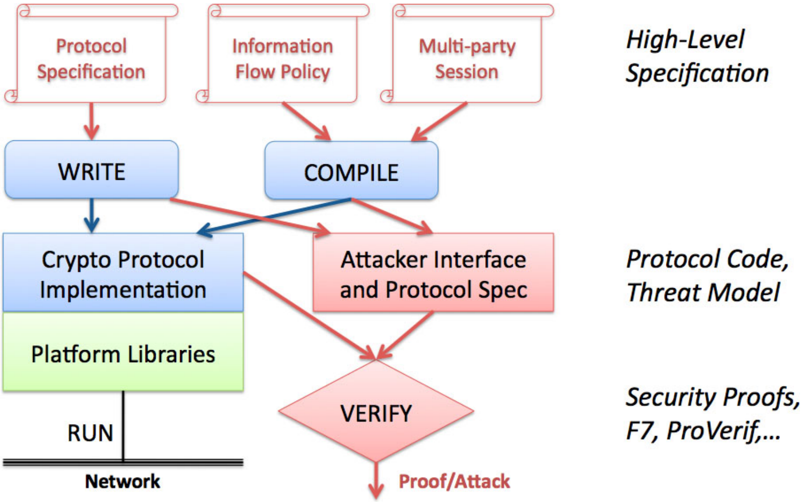
# Secure sessions

- passing authenticated (signed) values between 2 *run-times*
- design of a mini F# + primitives for authentication
  + global contract with sessions types
- compiling scheme into a low-level language ($\simeq$ pi-calculus)
  to describe authentication protocols
- formal proof of its correctness, with security property induced by strong typing of F# + usage of authentication primitives
- extension to other security properties, sessions V2
  (privacy of message values, integrity, dynamic number of principals, etc)

  [Corin, Deniélou, Leifer, Fournet, Bhargavan]

  [JCS 08, TGC 07, CSF 09, Deniélou phd 11]

# Example 3 Verified Crypto Protocol Implementations



# Protocol Specifications in F7

**$A \rightarrow B$: m, hmac $k_{AB}$ m**

## F# Code

```
let client a b k m =
  Pi.assume (ClientSent(a,b,m));
  let h = hmac k m in
  let msg = concat m h in
    Net.send msg

let server a b k =
  let msg = Net.recv in
  let (m,h) = iconcat msg in
    hmacVerify k m h;
    Pi.expect (ClientSent(a,b,m))
```

## F7 Interface

```
val client: a:prin -> b:prin ->
            k:key{SharedKey(a,b,k)} ->
            m:bytes -> unit

val server: a:prin -> b:prin ->
            k:key{SharedKey(a,b,k)} ->
            m:bytes{ClientSent(a,b,m)}
```

Dependent, refinement types

Pre-condition

Post-condition

# Protocol Specifications in F7

**$A \rightarrow B$: m, hmac $k_{AB}$ m**

## F# Code

```
let client a b k m =
  Pi.assume (ClientSent(a,b,m));
  let h = hmac k m in
  let msg = concat m h in
    Net.send msg

let server a b k =
  let msg = Net.recv in
  let (m,h) = iconcat msg in
    hmacVerify k m h;
    Pi.expect (ClientSent(a,b,m))
```

## Refined Crypto Interface

```
val hmac: k:key{MKey(k)} ->
          m:bytes{MayMAC(k,m)} ->
          h:bytes
```

Pre-conditions

```
val hmacVerify: k:key{MKey(k)} ->
                m:bytes -> h:bytes ->
                unit{MayMAC(k,m)}
```

Post-condition

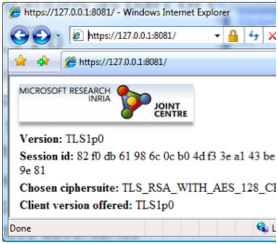
```
assume !a,b,k.
  SharedKey(a,b,k) => MKey(k)
assume !a,b,k.
  SharedKey(a,b,k) =>
    (!m. MayMAC(k,m) <=> ClientSent(a,b,m))
```

# TLS in F#

We implemented a subset of TLS (10 kLOC)

- Supports SSL3.0, TLS1.0, TLS1.1
  DES, AES, RC4, SHA1, MD5
- Largest verified crypto protocol implementation till date

We used "global" cryptographic verifiers, ProVerif and CryptoVerif [Blanchet]




We reached the limit of this proof method:

- Whole-program analysis does not scale
- Verification takes hours on a large machine

*Ongoing work:* Use F7 for modular verification

[CCS 08, TOPLAS 10, APLAS 10, POPL 10, ESORICS 09, phD Guts'11]

# Other works

- Acute – type safed marshalling [Leifer, Peskine, Zappa Nardelli]
- OTT – A semantics tool [Sewell, Zappa Nardelli]
- Scripting languages (Like types) [Zappa Nardelli]
- Jocaml (version 3; more portable, documentation) [Maranget, Mandel]
- Separation logic [Appel, Zappa Nardelli]
- Security through logs [Guts, Fournet, Zappa Nardelli]
- Information flow [Corin, Fournet, le Guernic, Planul, Rezk]
- Pattern-matching in Ocaml [Maranget]

# Miscellaneous

# Links

- Microsoft Research Cambridge through the MSR-INRIA Joint Centre
- Sewell et al at Cambridge, Computer Lab
- Indes, Celtique, PPS with ANR Parsec [Zappa Nardelli]
- Gallium for general discussion about programming languages
- Andrew Appel, Princeton
- Secsi, Cascade with ERC-Crysp [Bhargavan]

# Software

- diy tool suite [Alglave, Maranget]
- OTT: a semantics tool [Sewell, Zappa Nardelli]
- CompCertTSO: certified compiler for TSO [Jagannathan, Sewell, Sevcik, Vafeiadis, Zappa Nardelli]
- S2ML [Bhargavan, Corin, Deniélou]
- FS2CV [Bhargavan, Corin, Zalinescu]
- F7 [Bhargavan]
- Jocaml [Maranget, Mandel]
- 5% Ocaml (pattern matching) [Maranget]
- Hévéa: an efficient translator of Tex into Html [Maranget]

# Teaching

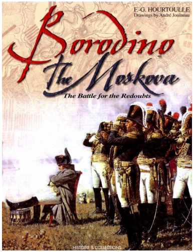
- MPRI (master course at Paris 7)
  `[Zappa Nardelli, Leifer]`
- École polytechnique
  `[Maranget, Bhargavan, ...Lévy (1992-2006)]`
  lecture notes + web pages
- Entrance examination at Polytechnique
  `[Maranget (4 years), Lévy (??-2009)]`
- Bertinoro, IIT-Delhi, Tsinghua, etc.

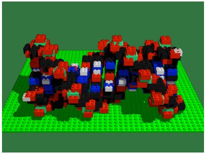
# Objectives for next years

# Scientific goals

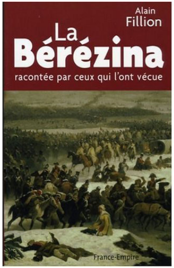
- Weak Memory Models
  - **ARM** multi-core + xfer to industry
  - **automatic** exploration of WMMs
  - automatic synchronisation of programs
  - certified compilation of C-like with **C1x/C++0x** WMM
- Security compilers and verifiers
  - **scalable** tools to verify security of programs
  - verified open source cryptographic **libraries**
  - **web** applications with formal proofs of security

# Organization

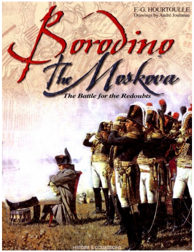


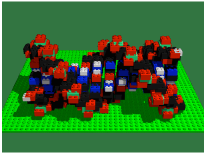



INRIA



# Organization



INRIA

# FIN

12 April 2011: http://msr-inria.inria.fr/forum2011