



Butterfly™

A Substrate for Distributed Computing

D.McClain

SpectroDynamics, LLC

BITA – Butterfly™ is the Answer

- Ideas shamelessly borrowed from Erlang, and then improved through the full power of Lisp
- Local and distributed computing
- Hot-loading of code fixes to remote servers (*FRUs*)
- Portable for Lispworks, Allegro, ClozureCL, SBCL
- Runs on OS X, Windows, Linux

BINE - Butterfly™ is not Erlang

- Erlang makes lightweight processes a fundamental computing unit; everything a process
- Erlang uses a ~~toy~~ very weak (*mostly-*) FPL
- Lisp processes are heavier
- Lisp is the most powerful language
- Do more in Lisp with fewer Processes
- Net massive gain with Lisp

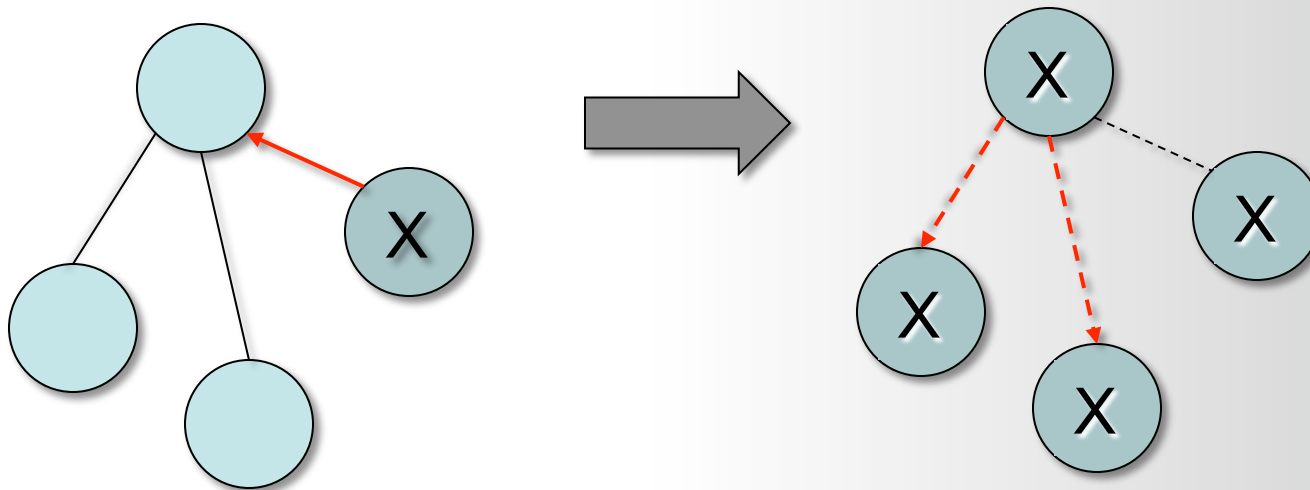
Butterfly™ & STM

- Software Transactional Memory system used for system table updates
- All-or-none Multi-Lock used for coordination of transactions with irreversible side effects – e.g., spawning and linking new processes
- High-performance mailboxes with selective receive
 - freedom to choose order of receipt greatly facilitates robust communications

Process Control

- **Process Instantiation**
 - (spawn fn) → pid
 - (spawn-link fn) → pid
- **Process Linking**
 - (link pid)
 - (unlink pid)
- **Process Termination**
 - (exit pid reason)
- (self) → pid of current process

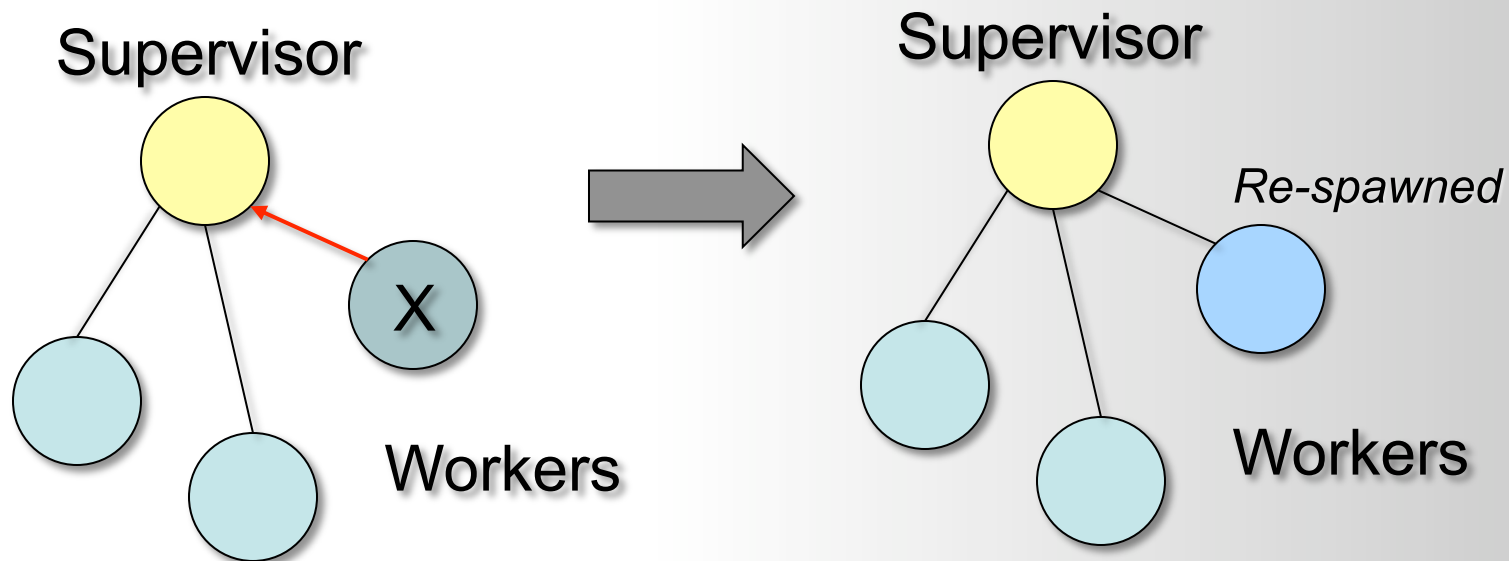
Processes and Links



Links are bidirectional

EXIT messages are broadcast to all linked processes

Supervisor Processes



Links are bidirectional

Supervisor intercepts EXIT and restarts failing PID

Interprocess Communication

- Interprocess primitives
 - (! pid msg) = send msg to pid
 - (!? pid msg) = rpc msg to pid, await response
 - (?? predicate) = selective receive
 - (RECV (msg) &rest pattern-clauses) = selective receive with pattern matching
- Standard Server Package
 - uniform message handling with FRU
 - user has only to code handlers, not message protocol
 - IPC primitives rarely used
 - Individual server packages provide API above IPC primitives

MATCH & RECV

- Powerful optimizing MATCH & RECV macros
- Can match against all primitive types, lists, vectors, arrays, structs, and #T templates for class instances, structs, and computed elements
- Merges longest common match prefixes, regardless of clause ordering, but retains local ordering
- Fully nestable

Match Example

```
(match x  
  ( ('A 'B 'C) (oper1))  
  ( ('A 'B 'D) (oper2))  
  ( ('A 'B va1) (oper3 va1)))
```

'A 'B 'C and 'D are symbol constants
va1 is a match variable

Compiler efficiently groups clause predicates
using longest common prefix matching

Compiled Match Example

```
(LET ((#:ANS5634
      (SYMBOL-MACROLET
        ((#:LBL5635 (LAMBDA () (OPER1)))
         (#:LBL5636 (LAMBDA () (OPER2)))
         (#:LBL5637 (LAMBDA (VAL) (OPER3 VAL))))
      (BLOCK #:BLOCK5633
        (LET ((#:ARG5632 X)
              (WHEN (CONSP #:ARG5632)
                (WHEN (EQ (CAR (THE CONS #:ARG5632)) 'A)
                  (LET ((#:TL5639 (CDR (THE CONS #:ARG5632))))
                    (WHEN (CONSP #:TL5639)
                      (WHEN (EQ (CAR (THE CONS #:TL5639)) 'B)
                        (LET ((#:TL5641 (CDR (THE CONS #:TL5639))))
                          (WHEN (CONSP #:TL5641)
                            (LET ((#:HD5642 (CAR (THE CONS #:TL5641))))
                              (WHEN (EQ #:HD5642 'C)
                                (UNLESS (CDR (THE CONS #:TL5641))
                                  (RETURN-FROM #:BLOCK5633 (LIST #:LBL5635))))
                              (WHEN (EQ #:HD5642 'D)
                                (UNLESS (CDR (THE CONS #:TL5641))
                                  (RETURN-FROM #:BLOCK5633 (LIST #:LBL5636))))
                              (UNLESS (CDR (THE CONS #:TL5641))
                                (RETURN-FROM #:BLOCK5633
                                  (LIST #:LBL5637 #:HD5642))))))))))))
              (MATCH-FAIL))))))
      (DECLARE (CONS #:ANS5634))
      (APPLY (CAR #:ANS5634) (CDR #:ANS5634))))
```

Examples

Local Service -- or -- Server Local Code

```
(register-service :ECHO  
  (spawn #'echo-server))
```

-- or -- Remote Service for Client

```
(register-service :ECHO  
  (remote-service "Artemis.local" :name :ECHO))
```

Client Code

```
(!? :ECHO "Hello") → "Hello"
```

Standard Server Example

```
(defun echo-service (state server-loop-fn)
  (funcall server-loop-fn 'echo-handler state))

(defun echo-handler (msg state)
  (make-handler-response
   :answer msg
   :state state))

(register-service :ECHO
  (make-server 'echo-service)))
```

System-wide Services

- **Exit-Monitor**
 - logs all EXIT notifications
- **Keep-Alive Service**
 - generic supervisor to restart valued services
 - settable heuristics for restarts
 - default heuristic = “should not restart service if we see too many failures (>5) during one-minute period – might be code bit-rot”
- **System-Log**
 - rotating log file management
 - provides permanent log of whatever is LOG'd
 - provides background stream printing for user monitoring

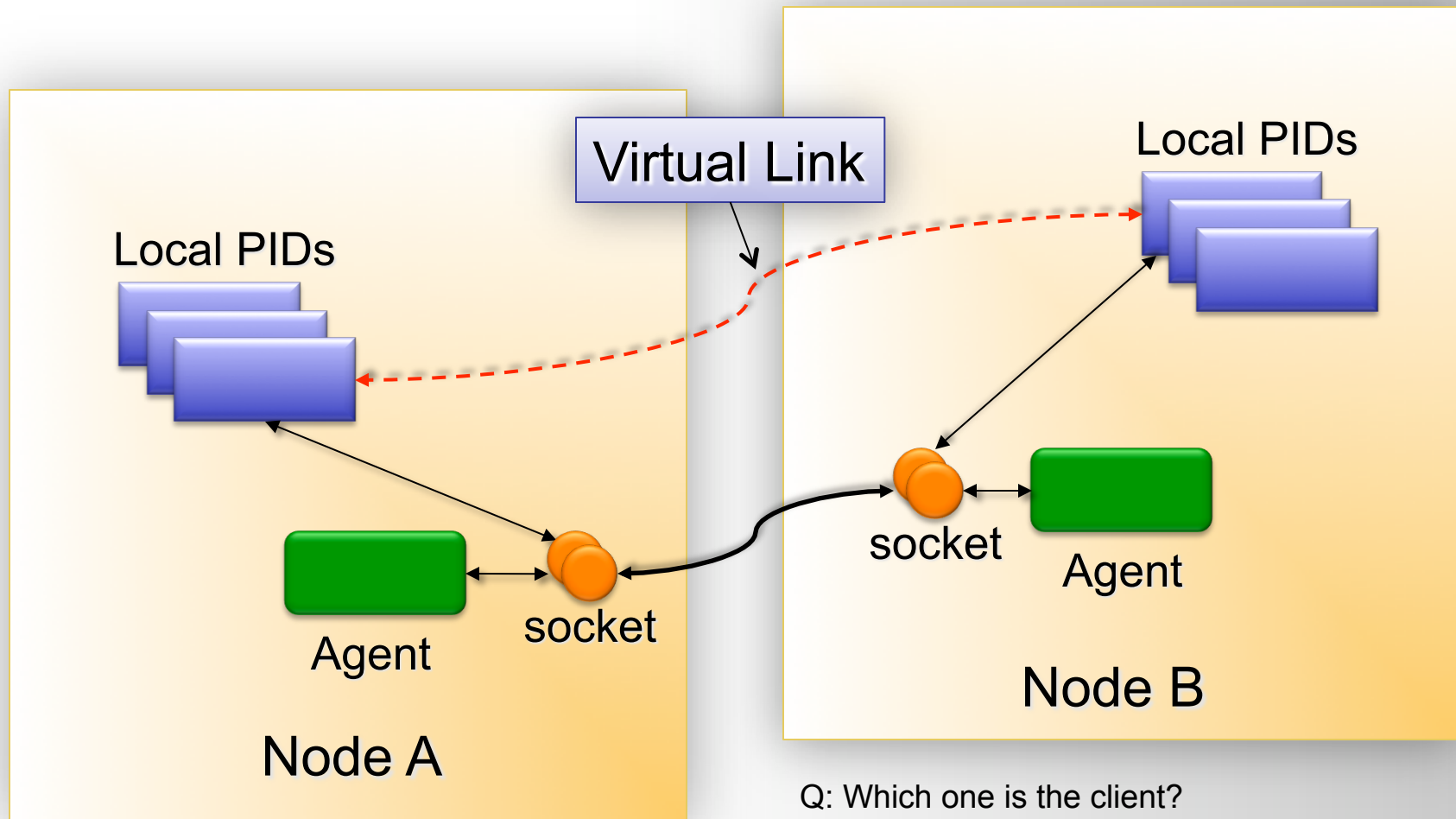
Remote Processes

- Remote PID's are recognized, and messages are relayed through node sockets
 - resolution of remote PID's is transparent to user code
 - the same primitives (!, !?, ??, RECV) are used regardless of whether the target PID is local or remote
 - services code can be transparently and dynamically redeployed in any manner of distribution across nodes

Domain Resolution

- (REGISTER-SERVER name node)
 - adds node IP address to server information table
 - Node examples:
 - “artemis.local:12010”
 - “192.68.1.101:12010”
- (REGISTER-SERVICE name pid)
 - adds pid to service information table
- PID - actual PID object, or symbolic representation resolved through tables

Remote Butterfly™



Message Contents

- Local Services
 - any Lisp object
- Messengers
 - marshal and unmarshal data with a network neutral persistent encoding
 - manage socket interfaces
 - act as proxy targets and perform message forwarding
- Remote Services
 - any portable Lisp object
 - excludes compiled code and runtime closures
- Agents act on remote client's behalf to manage requests for remote spawn, link, unlink, exit

Butterfly™ Security

- Butterfly™ uses high-security network communications
 - Because BFLY carries sensitive database information...
 - Because BFLY allows clients to execute nearly anything on a remote host...
 - Because BFLY allows hot-loading of code patches to running code...
 - Because BFLY is expected to operate in wireless networks...
- Butterfly™ uses stringent network security measures:
 - Initial connection is established using a two-way handshake with a SHA256 digest
 - Handshake is a challenge / response from server to client, followed by a challenge / response from client to server
 - Failure during challenge / response shuts down the network connection
 - All data transmissions are encrypted using asymmetric AES128 with roving keys
 - Different encryption keys for client-to-server and server-to-client
 - Even the handshakes are encrypted
 - All of this on top of serialization encoding
 - Explicitly NOT using OpenSSL