Concurrent Objects

-- Introspect, Extrospect & Prospect --

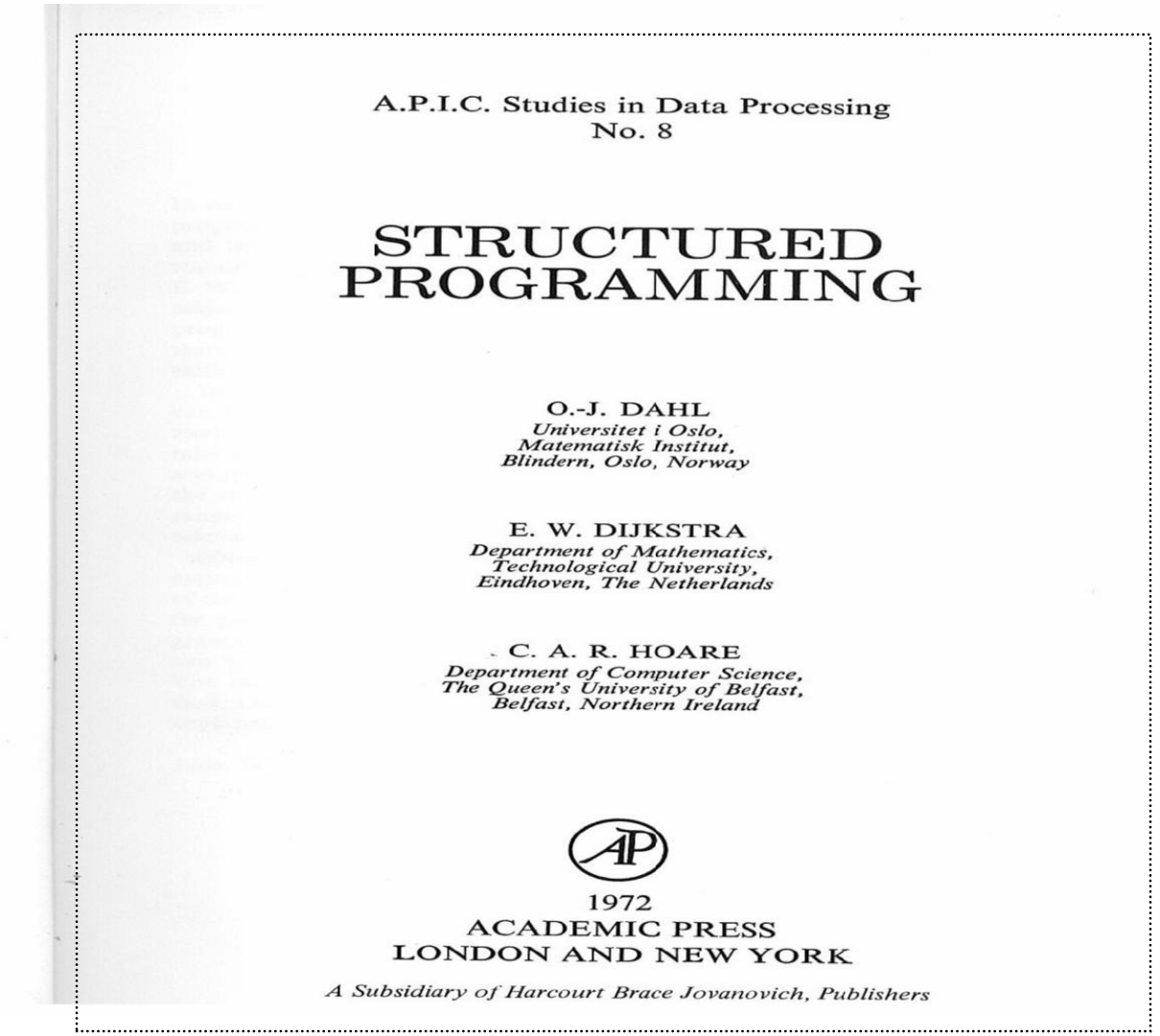
Aki Yonezawa
Dept. of Computer Science &
Information Technology Center
University of Tokyo

Plan of Talk

- Background
  - How I came up with the idea - *Introspect*
- Tetrahedron of Language Research – *my tenet*
  - Computational Reflection
  - Linear Logic Semantics
  - Implementations on Massively Parallel Machines *(Introspect & Prospect)*
  - Mobile Concurrent Object - *JavaGo*
  - Applications:
    - N-body, Space-Station,…
- Massive Use of Concurrent Objects
  - Linden's Second Life - *Extrospect*
- *Prospect*

Thank to B.Liskov's Lecture (1974)

Dahl's Book



A.P.I.C. Studies in Data Processing
No. 8

STRUCTURED PROGRAMMING

O.-J. DAHL
Universitet i Oslo,
Matematisk Institut,
Blindern, Oslo, Norway

E. W. DIJKSTRA
Department of Mathematics,
Technological University,
Eindhoven, The Netherlands

C. A. R. HOARE
Department of Computer Science,
The Queen's University of Belfast,
Belfast, Northern Ireland

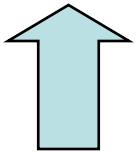
1972
ACADEMIC PRESS
LONDON AND NEW YORK
A Subsidiary of Harcourt Brace Jovanovich, Publishers

Thank to C.Hewitt's suggestion (1976)

Nygaard's Book

after Simula67, and Early 70's

- Smalltalk – 1972 language interface to dynabook
- CLU (abstract data types) - 1973
- Minsky's Frame - 1974
- Hewitt's Actor – 1973 universal modular forms for AI
- Capability-based OS - 1975
- Entity-Relationship Model – 1973 data model



Structuring and Modularizing programs and knowledge representations

Both Modeling and Programming

Early 70s in Tokyo, I worked on languages and theorem proving. Then, I started being interested in:

1. **modeling worlds and simulate them on computers!**

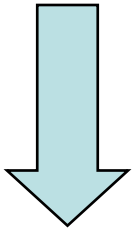
2. **powerful programming frameworks!**

Our goal in programming research & OO

- **Reducing the complexity of software systems, while maintaining reasonable performance**
- **Making**

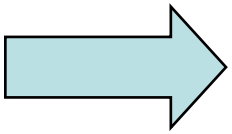
  software systems and software construction

  simpler and more manageable



enabling construction of

more powerful software systems

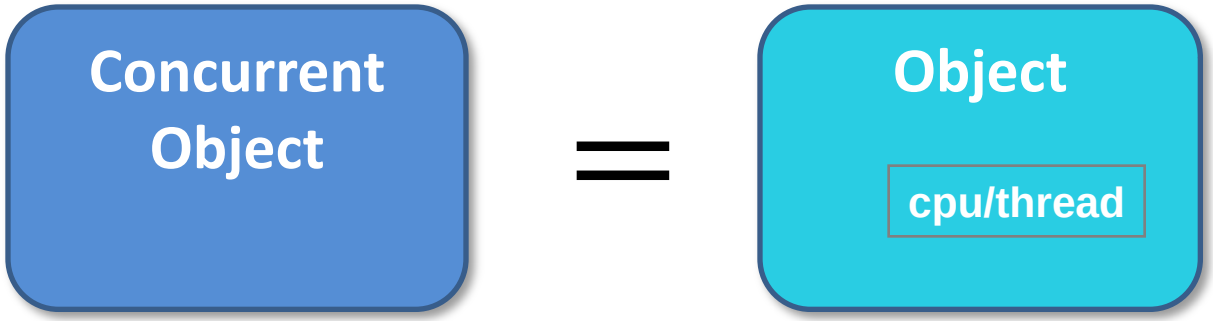


Object Orientation emerged!!

My idea formed around 1974

For modeling and programming,…

- **Concurrent Object**
  = Encapsulated(Stateful Object + thread)

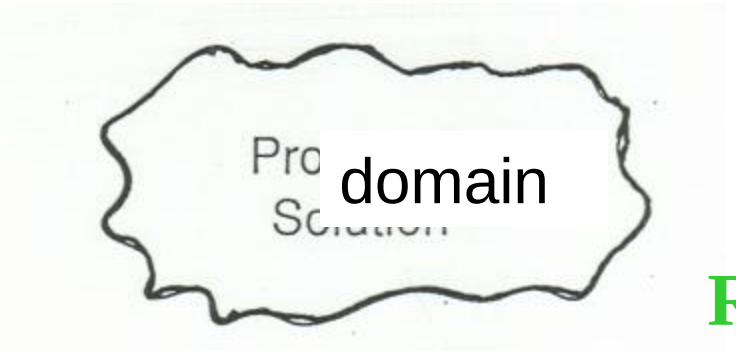


- **Asynchronous message passing among concurrent objects**

- **Different Approach**

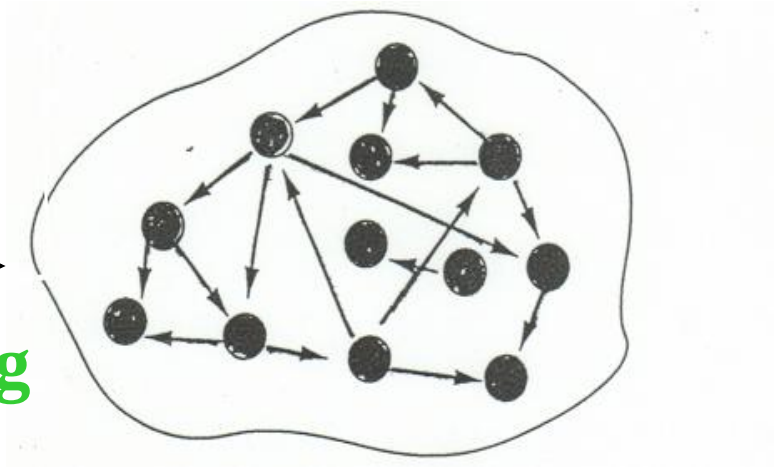
-C.Hewitt and H.Baker:
***Laws for communicating Parallel Processes*, IFIP1977**
-G.Agha: *A Model for Concurrent Computation in Distributed Systems*, MIT Press 1987

my Modeling of Real World in Concurrent Objects

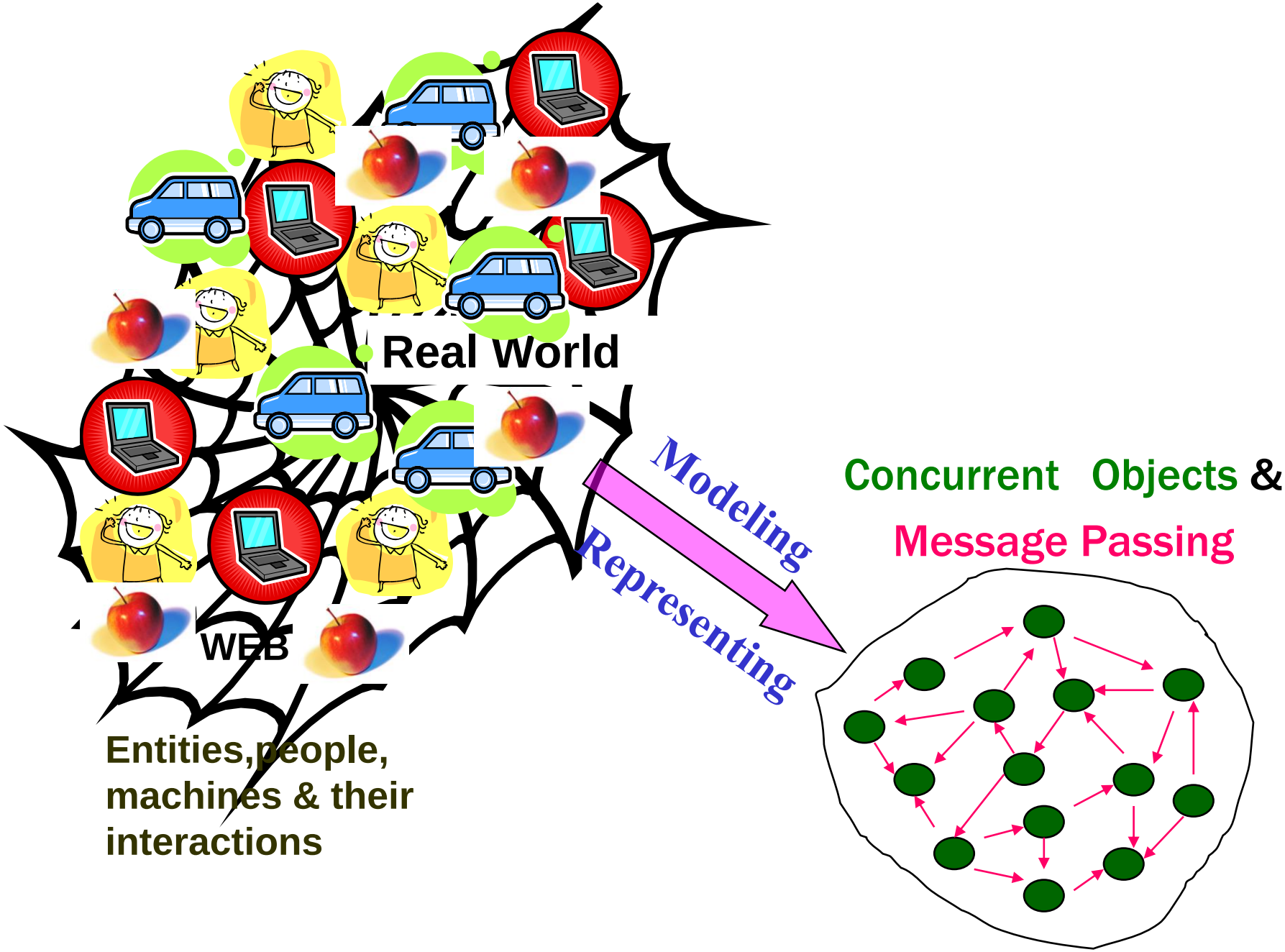


Modeling

Representing



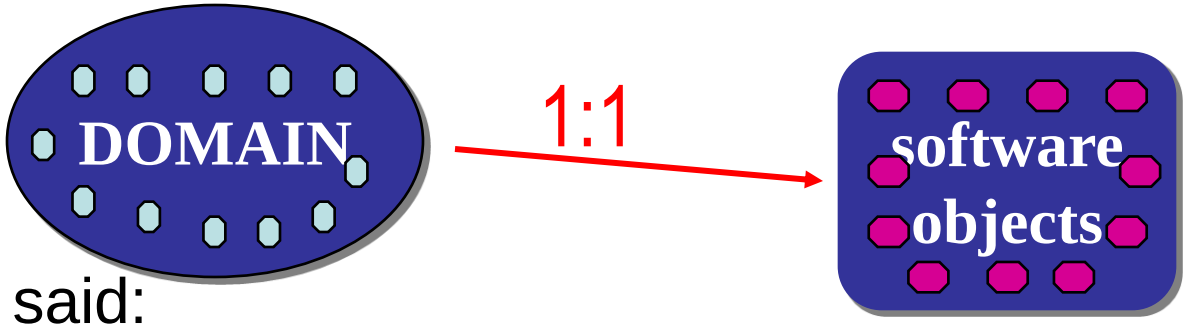
concurrent


Real World
WEB
Entities,people,
machines & their
interactions
Modeling
Representing
Concurrent Objects &
Message Passing

Natural Modeling of a World

- **Natural** modeling **reduces** complexity
- **Naturalness means directness!**
  - **1:1 mapping**

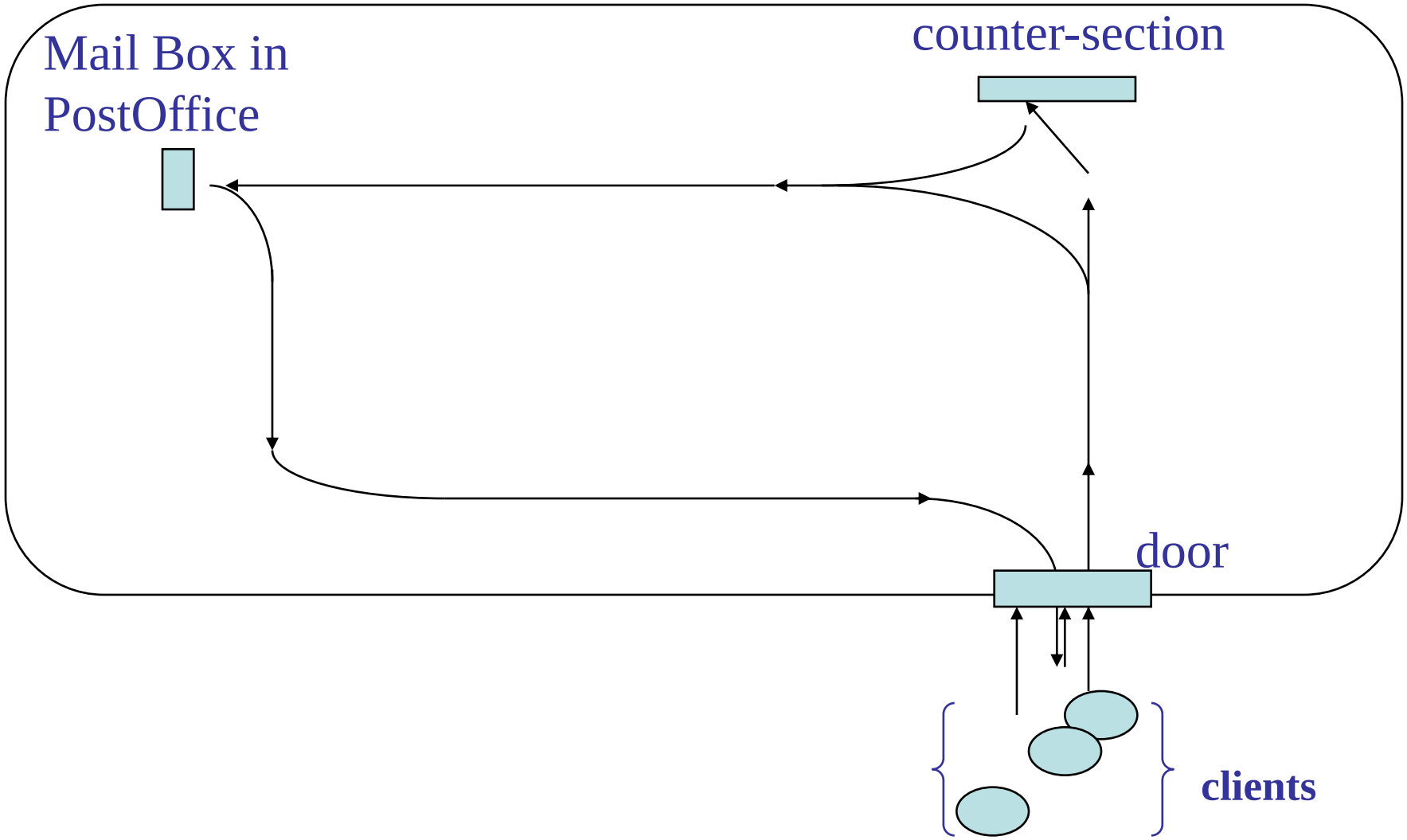
    from domain objects to software modules



- Ole Madsen said:
  - Objects and Classes are **well-suited** for modeling **physical entities** and associated concepts
  - **"Concurrency" is *MUST* for modeling**

Example:

Modeling a Post Office (’77)



Modeling Post Office in COs

- Post Office Building ⇔ the door
  - ➔ door **concurrent object**
- Counter with clerks
  - ➔ counter **concurrent objects**
- Mail Box
  - ➔ mailbox **concurrent object**
- Customers
  - ➔ customer **concurrent objects**

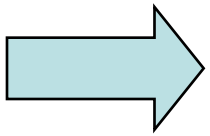
not messages!

Modeling *Movement* of Customers

- **Two ways:**

  1. a customer object is transmitted in a message

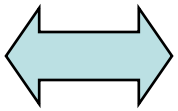
  2. a customer object moves by itself



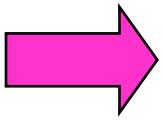
Object (or its code) migrates!!

Learning from Ambient

- **Non-local customers do not know the internal geography of the local post office.**



Customer object does not know the location/name of counter objects



Customer objects must learn the location of the counter object from the Door object

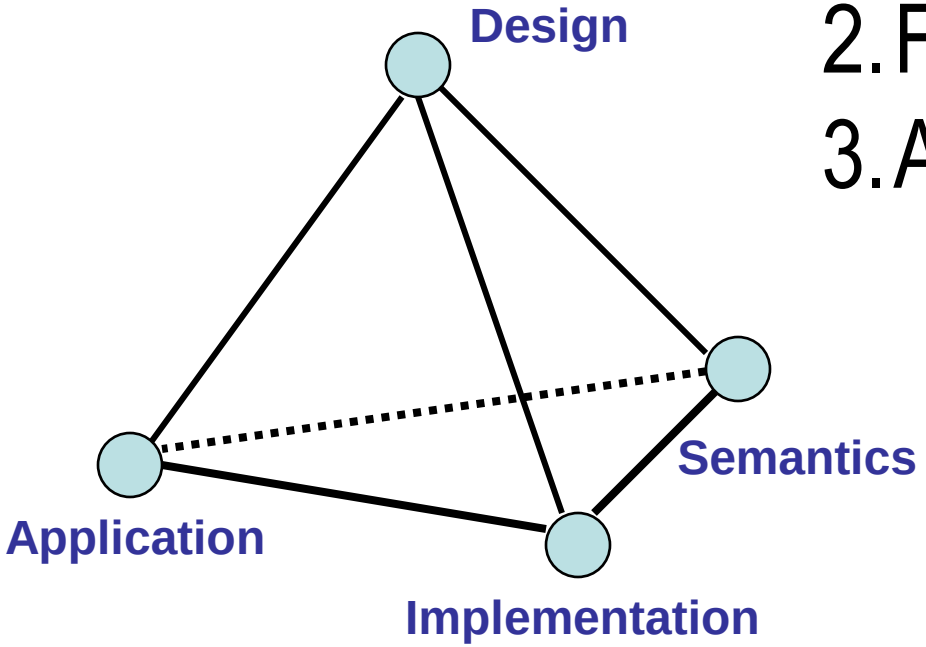
ambient

Tetrahedron of my Language Research

When you design a language, then

1. Implementation
2. Formal Semantics
3. Appli/Programming

must be worked out!

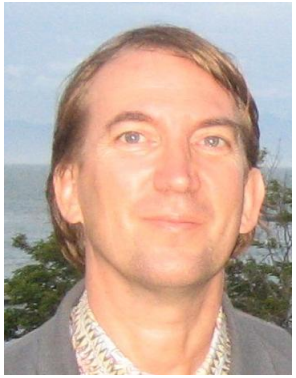


Overview of my Research since 1984

- Language Design
  - ← *ABCL* language series (JP Briot, Shibayama) 1984-
  - ← Inheritance Anomaly (S.Matsuoka, JP Briot) 1985-1989
  - ← Reflection (T.Watanabe) 1988
- Semantics
  - ← fragment of Linear Logic (N.Kobayashi) 1991-
- MPP Implementations
  - ← StackThread scheme (K.Taura) 1993-
- Mobile objects and its implementation
  - ← JavaGo (T.Sekiguchi, H.Masuhara) 1999
- Appli/Programming
  - ← N-body, Space station dynamics, CFG-parser… 1997

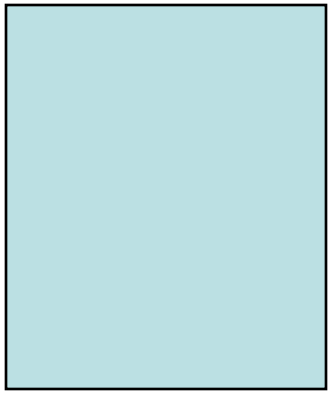
Collaboraters

E.Shibayama J-P. Briot S.Matsuoka N.Kobayashi

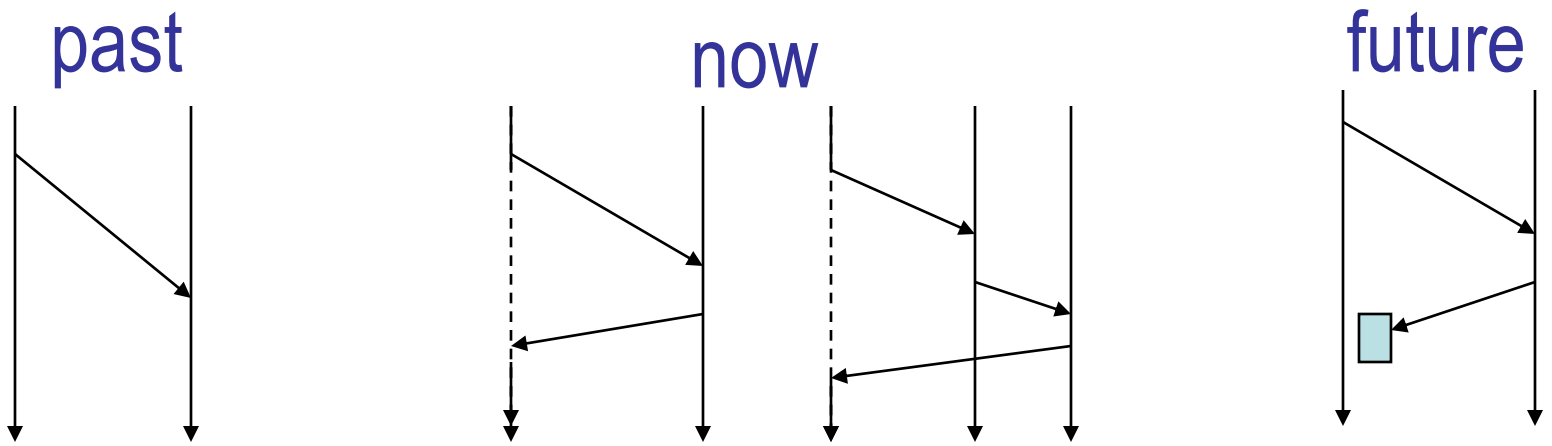
K.Taura H.Masuhara T. Watanabe T.Sekiguchi



Message Passing in *ABCL/1*

- Message passing is asynchronous.
  - more natural and more parallelism
- Three types of message transmissions:
  - Send-and-no-block (past)
  - Send-and-wait-for-reply (now)
  - Send-with-future (future)



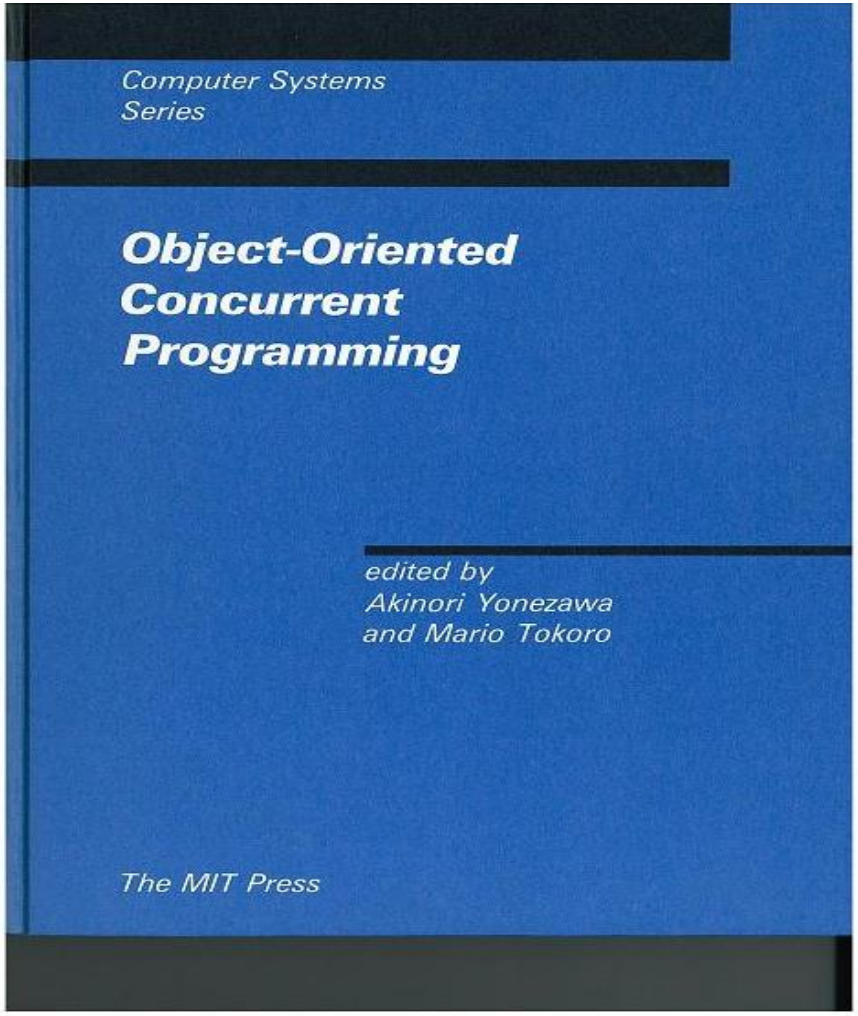
Our First Language ABCL/1 (1984)

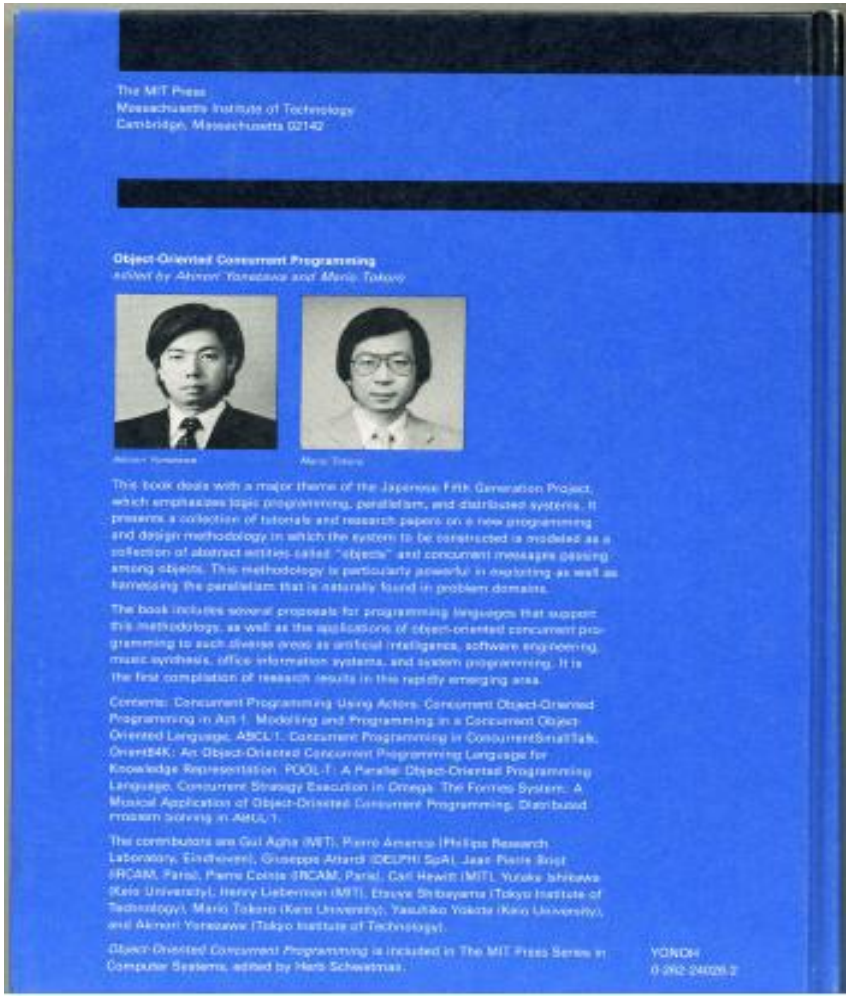
- First concurrent object-oriented language..
- Each CO (concurrent object) has a single thread.
- At any time, a CO is in one of three modes:

  (1) *dormant,* (2) *active,* (3) *waiting*

- No inheritance

Book in 1987


Computer Systems
Series
Object-Oriented
Concurrent
Programming
edited by
Akinori Yonezawa
and Mario Tokoro
The MIT Press


The MIT Press
Massachusetts Institute of Technology
Cambridge, Massachusetts 02142
Object-Oriented Concurrent Programming
edited by Akinori Yonezawa and Mario Tokoro
This book deals with a major theme of the Japanese Fifth Generation Project, which emphasizes logic programming, parallelism, and distributed systems. It presents a collection of tutorials and research papers on a new programming and design methodology in which the system to be constructed is modeled as a collection of abstract entities called "objects" and concurrent messages passing among objects. This methodology is particularly powerful in exploiting as well as harnessing the parallelism that is naturally found in problem domains.
The book includes several proposals for programming languages that support this methodology, as well as the applications of object-oriented concurrent programming to such diverse areas as artificial intelligence, software engineering, music synthesis, office information systems, and system programming. It is the first compilation of research results in this rapidly emerging area.
Contents: Concurrent Programming Using Actors. Concurrent Object-Oriented Programming in Act 1. Modeling and Programming in a Concurrent Object Oriented Language, ABCL/1. Concurrent Programming in ConcurrentSmallTalk. OrientB4K: An Object-Oriented Concurrent Programming Language for Knowledge Representation. POOL-T: A Parallel Object-Oriented Programming Language. Concurrent Strategy Execution in Omega. The Fermis System: A Musical Application of Object-Oriented Concurrent Programming. Distributed problem solving in ABCL/1.
The contributors are Gul Agha (MIT), Pierre America (Philips Research Laboratory, Eindhoven), Giuseppe Attardi (DELPHI SpA), Jean Pierre Briot (IRCAM, Paris), Pierre Cointe (IRCAM, Paris), Carl Hewitt (MIT), Yutaka Ishikawa (Keio University), Henry Lieberman (MIT), Etsuya Shibayama (Tokyo Institute of Technology), Mario Tokoro (Keio University), Yasuhiko Yokote (Keio University), and Akinori Yonezawa (Tokyo Institute of Technology).
Object-Oriented Concurrent Programming is included in The MIT Press Series in Computer Systems, edited by Herb Schwetman.
YONOH
0-262-24026-2

H.Lieberman

G.Agha & C.Hewitt

P.Cointe

P.America

G.Attardi

Contents

Implementation and Applications of ABCL/1

- A Lisp-based implementation on SUN ws.
- Manual and programming guide were distributed in OOPSLA'86.
- A more complete implementation on Lisp machine in 1987.
- CO based parser for Context Free Grammar
  - English grammar with 250 no-terminal symbols in 1987.
  - A popular paper published in Coling'88 (Computational Linguistic Conference in Budapest, 1988)

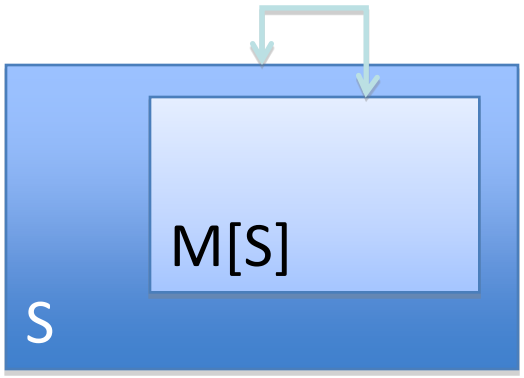
Concurrent OO *Reflection*

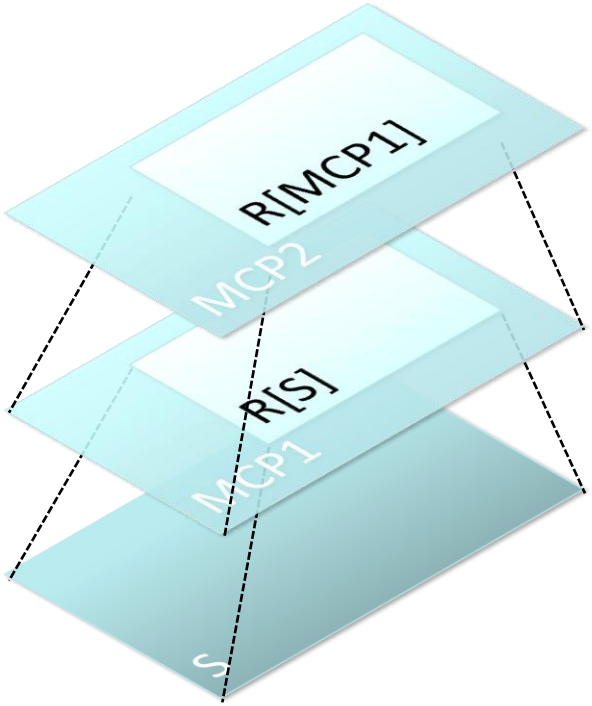
- Inspired by B. Smith, 3-Lisp
- Inspired by P.Maes and L.Steels, 3-KRS

  – With Takuo Watanabe

Computational Reflection

Computation about Oneself: Introspection & Self Modification





MCPs: meta-interpreters, meta-objects

base-level

A reflective system S can reason about or act upon itself via the causally-connected self-representation M[S]=Model of S.

Pioneers:
3-Lisp (B. C. Smith, 1982)
3-KRS (P. Maes, 1986)

Representation in a Reflective Tower (Smith, 1982)
S is reified as R[S] within the meta-circular processor MCP1. MCP1 is also reified in MCP2, and so forth. Reflective behaviors are realized as normal operations in the meta-levels (MCPs).

ABCL/R

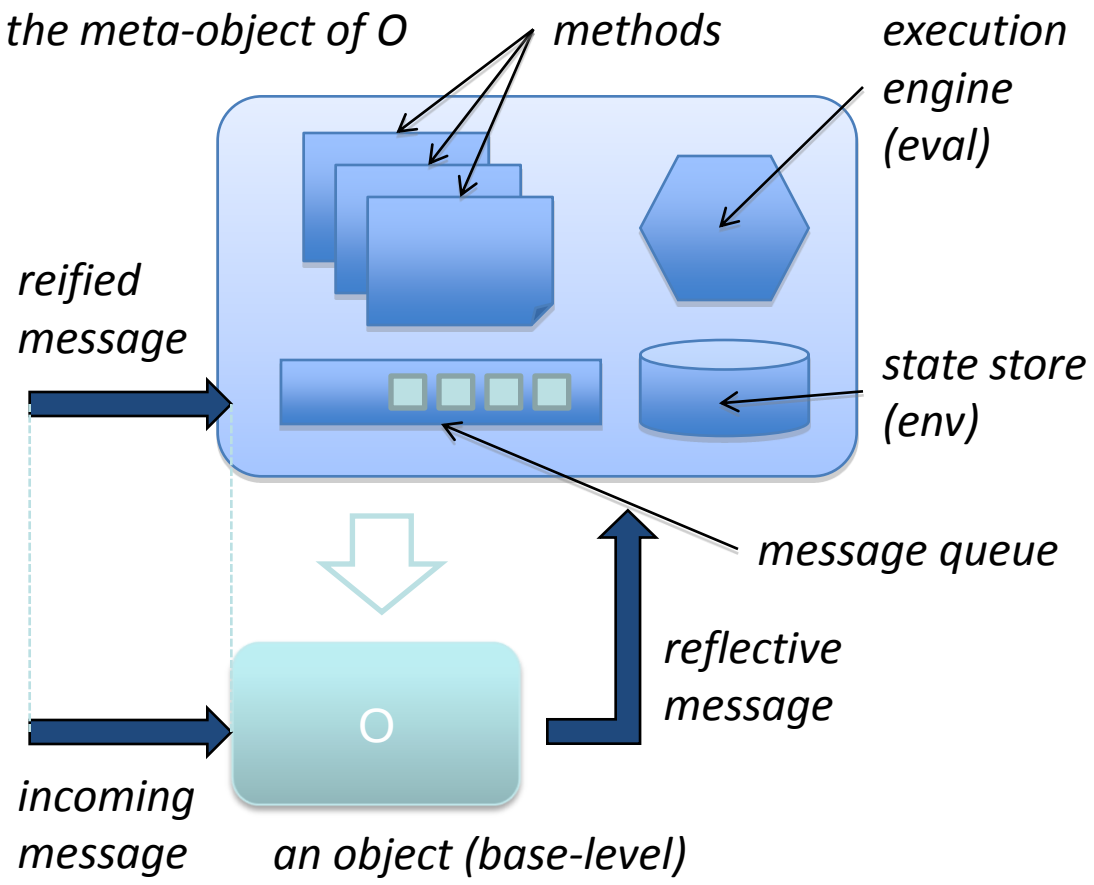
Decided to write an interpreter of CO!

One Concurrent Object A –> One Meta-Object (Model of A)

Each concurrent object has its own meta-object that reifies its entire structure and solely governs its computation.

The meta-object is a 1st class object and thus has its meta-object. This implies that the reflective tower exists for every object.

Any object can send messages to its meta-object. Reflective behaviors are realized with such inter-level messages.

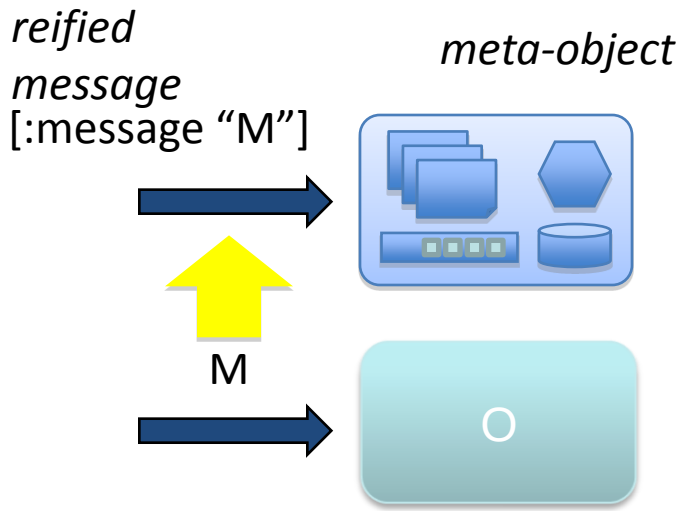


Watanabe & Yonezawa, OOPSLA '88

How the Meta-Object Works (1)

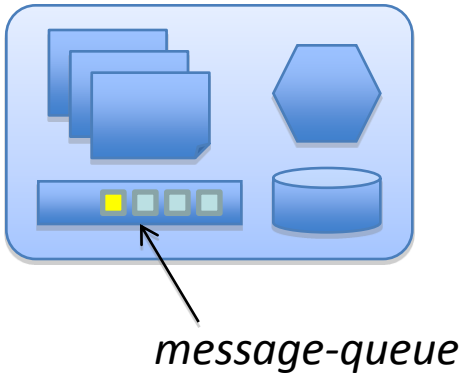
The Metacircular Interpretation of Concurrent Objects

(1) Suppose that an object O has just received a message M. This is interpreted as a reception of the reified message [:message “M”] by the meta-object of O.



(2) On receiving the reified message, the meta-object simply put it into its incoming message queue. Then set its execution mode to active.

```
[queue <== [:put ReifiedMessage]]
[mode := active]
```

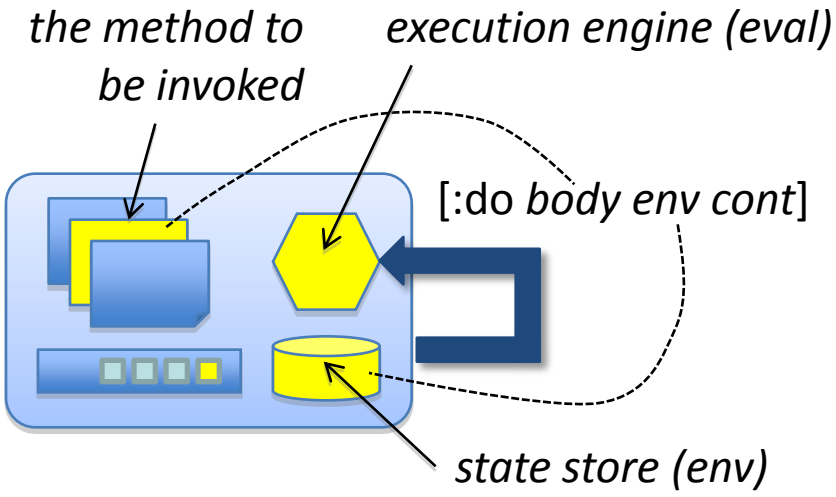


How the Meta-Object Works (2)

The Metacircular Interpretation of Concurrent Objects

(3) The active mode meta-object retrieves a message from the queue and looks up an appropriate method for it.

```
[msg := [queue <== :get]]
[mth := [methodpool <== [:lookup msg]]]
```



(4) The meta-object then starts invoking the method by sending a request to the execution engine (eval) object.

```
[eval <= [:do (body-of mth) env cont]]
```

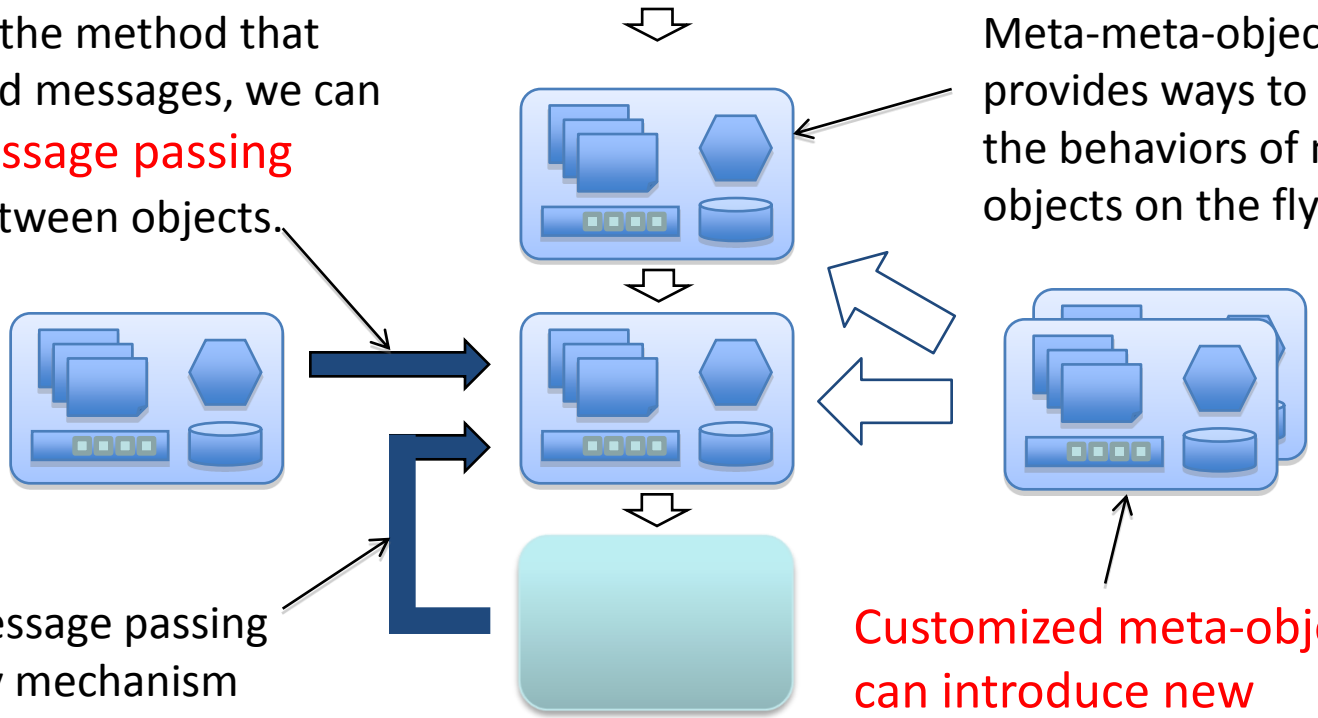
The message to the eval object contains the code, environment and continuation.

(5) The meta-object repeats the above actions while the queue has outstanding messages. When the queue gets empty, the object becomes dormant.

Use of Meta-Objects

Reflection allows Parts and Message Handler of a CO to be modified!!

By modifying the method that handles reified messages, we can add new message passing protocols between objects.



Meta-meta-objects provides ways to change the behaviors of meta-objects on the fly.

Inter-level message passing is the primary mechanism for providing explicit reflective behaviors.

Customized meta-objects can introduce new language features or modified object semantics.

Applications of ABCL/R

- Dynamic Acquisition of Methods
  - A simple example of inter-level messages
- On-the-fly Object Monitoring
  - Meta-meta-objects are used to add/remove monitoring in/out-messages in meta-objects.
- Modular Implementation of the Time-warp Algorithm
  - Customized meta-objects provide an encapsulated implementation of the algorithm.
    - Timewarp Algorithm: an optimistic algorithm for distributed discrete event simulations (by D. Jefferson, 1985)

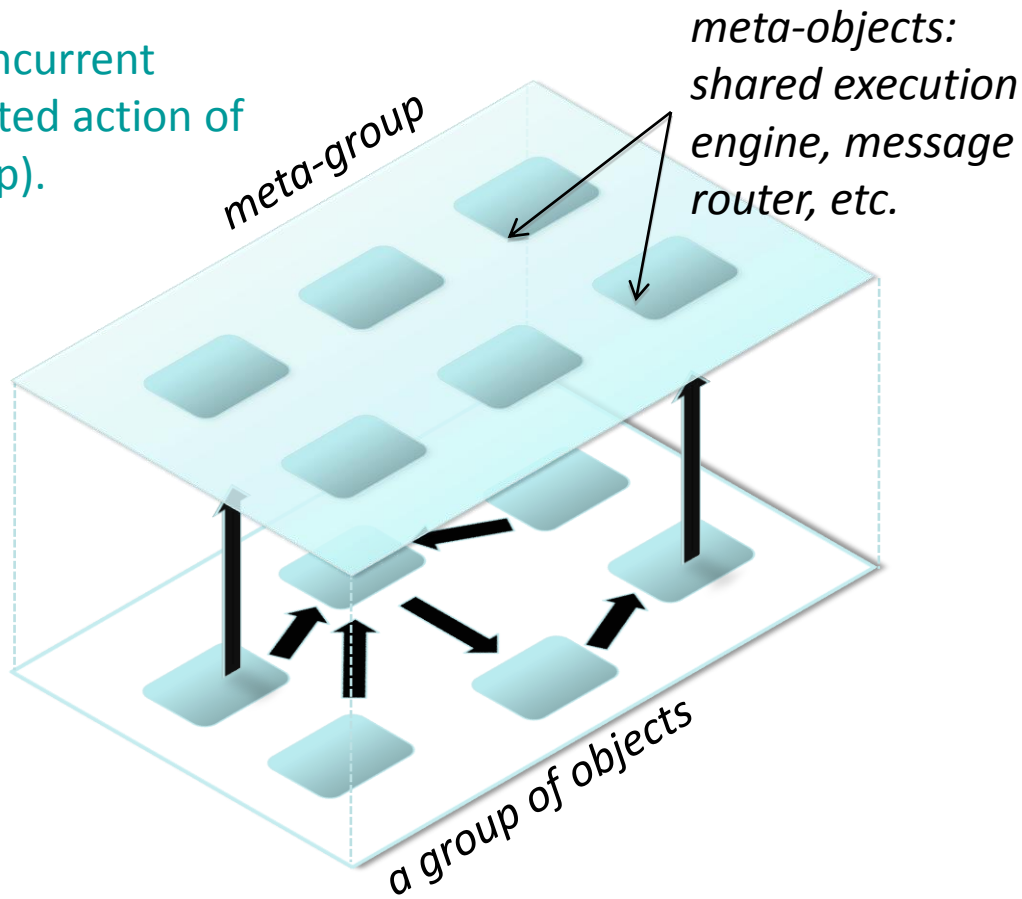
Group-Wide Reflection

A Collective Meta-Level for a Group of Concurrent Objects

Collective behavior of a group of concurrent objects is represented as a coordinated action of a group of meta-objects (meta-group).

The default behavior of meta-group is proved to simulate the behavior of base-level objects.

Reflective behaviors are realized by inter-level messages.

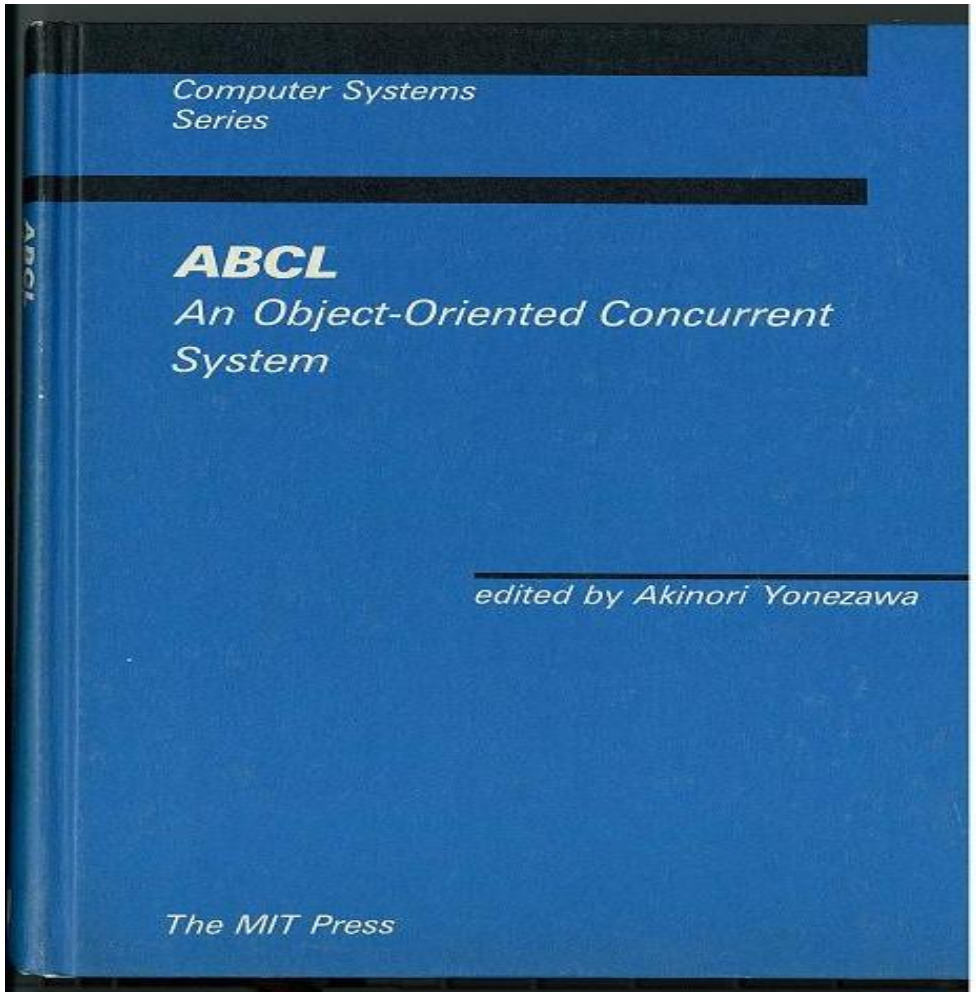


Applications:
Dynamic Object Migration,
Adaptive Scheduling, etc.

Watanabe & Yonezawa, REX/FOOL '90 (LNCS #489)

Book in 1990

- A collection of
our papers upto 1989,
Including "reflection",
"CFG parser", debugger,
language manuals etc.
- Excluded are
implementations:
  1) StackThreads
  2) JavaGo
and formal semantics



Linear Logic Semantics

- Wanted have formal/mathematical semantics for Concurrent OO Languages
- R. Milner's π-calculus was a choice…
  - British Empire of π-calculus was a bit…
  - Familiar with Gentzen's sequent style logic

⇒

Girard's Linear Logic was my choice!

Semantics for Concurrent Object Language


N.Kobayashi

- Project started in 1991
- Goals:
  - Formal foundations for concurrent object-oriented languages, to be used for:
    - language design, ***including type systems***
    - justification of compiler optimizations
    - program verification
    - ***research prestige***

Linear Logic

- **Resource-conscious logic [Girard 87]**

  A –o B ***linear implication***

  B can be obtained by consuming A

  A ⊗ B ***tensor product***

  A and B are available simultaneously

  A & B

  A and B are available, but not both (you have to choose one of them)

  !A

  An unbounded number A is available

Essence of Linear Logic

- **Example**
  - **A: one dollar**
  - **B: a coke (of one dollar)**
  - **C: a chocolate (of one dollar)**

A –o B **valid**

A –o C **valid**

A –o B⊗C **invalid**

(you cannot buy both with one dollar!)

A –o B&C **valid**

(you can buy whichever you like)

Linear Logic Formulas as Concurrent Objects

- *m -o A*
  - **An object that *receives/consume* message m, and then behaves like A**
- *m ⊗ A*
  - **An object *sends* message m, and then behaves like A**
- *Computation as deduction*

(c.f. logic programming)

$$\underbrace{(m \otimes A)}_{\text{sender}} \otimes \underbrace{(m \multimap B)}_{\text{receiver}} \multimap A \otimes B$$

Counter Objects as Linear Logic Formulas

$$!\forall n,inc,read.$$

(counter (n, inc, read) -o

$\forall$reply.(inc (reply) -o

(counter (n+1, inc, read) $\otimes$ reply()))

&

$\forall$reply.(read (reply) -o

(counter (n, inc, read) $\otimes$ reply(n))))

Types for Concurrent Objects
[OOPSLA 1994]

- Formula types as process types
  - O

    Type of formulas
    ≈ Type of objects and messages
  - int $\rightarrow$ O

    Type of predicates on integers
    ≈ Type of communication channels that carry integers
    e.g. $\forall$x:int.(c(x) –o d(x+1))
  - (int$\rightarrow$ O) $\rightarrow$ O

    Type of predicates on predicates on integers
    ≈ Type of communication channels that carry channels of type int$\rightarrow$ O
    e.g. $\forall$x:int $\rightarrow$ O.(c(x) –o x(1))

References

[1] Kobayashi and Yonezawa, *Asynchronous Communication Model based on Linear Logic*, Formal Aspects of Computing, 1995. (rec. by R. Milner)

(Basic computation model based on linear logic + encoding of actors, CCS, etc.)

[2] Kobayashi and Yonezawa, *Towards Foundations of Concurrent Object-Oriented Programming –Types and Language Design,* Theory&Practice of Object Systems, 1995

(Typed higher-order computation model based on higher-order linear logic + design of typed concurrent OO language on top of it)

Serious Implementations

on Massively Parallel Machines

– With Kenjiro Taura, Univ. Tokyo & S. Matsuoka

ABCL on Fujitsu AP1000 (1992-)

- Developed series of implementations of concurrent object-based languages on massively parallel machines (MPP).
- Intended for high performance computing.

One of the earliest attempts for *high performance* parallel language on distributed memory MPPs

[ACM PPoPP'93, ACM PLDI'97, ACM PPOPP'99]

In 1992,

- Variety of directions/beliefs in processor architecture
  - Dataflow: *T, EM4, J-Machine
  - MPPs: AP1000, CM5, -- ccNUMA: DASH
- Variety of original programming languages
  - OO: ABCL, Concert, …
  - Functional: Multilisp, Id, Sisal -- Logic: KL1
- We picked up our own language, ABCL/f to implement!!

What we have investigated

- Execution model of concurrent objects is:
  - *"objects, each with its own thread, are exchanging messages"*

- This could be literally implemented as:
  - *concurrent object = data + a thread of control*
- But this simply doesn't work with overwhelming resource usage of *threads*.

usual threads libraries provide

Ideas tested

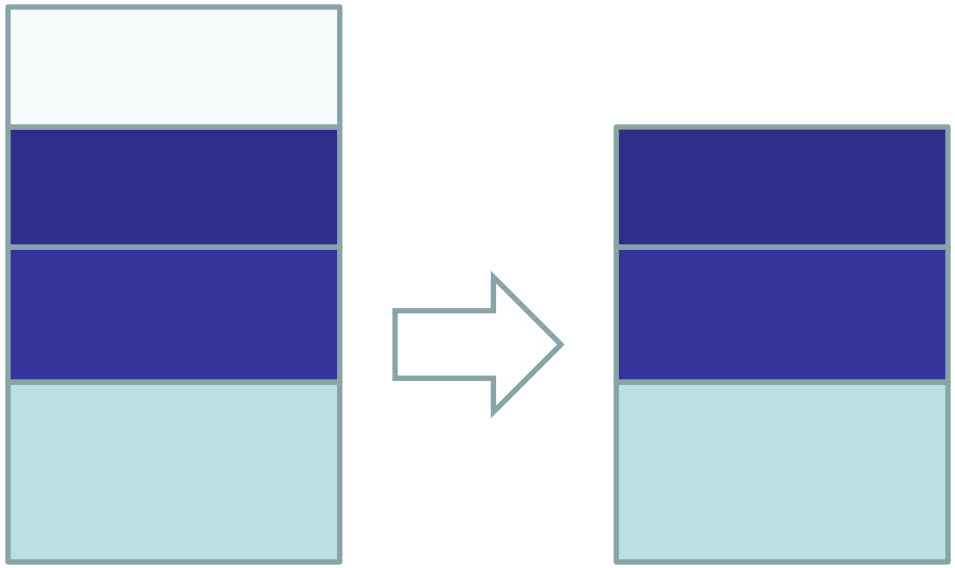
- Attempt 1: what's known as "thread pool"
  - Better than nothing, but the effect is limited
- Attempt 2: associate a thread with "asynchronous methods", not "concurrent objects"
  - Still too many threads with millions async. calls
- Attempt 3: "StackThreads" approach
  - *Speculatively execute all threads with one stack*

"StackThreads", our Approach

- Exec all threads on a single stack
- But how to "switch" between threads?
  - Simple! Manipulate intra-stack pointers, and remove the thread's frame from top of the stack.
- Very cheap & fast threads obtained!!

=>

A huge number of fine-grained threads is now usable.



StackThreads (cont'd)

- Reimplemented with a regular GNU C compiler as a backend (PLDI '97)
- Extended to shared memory multiprocessors with work stealing (PPoPP '99)
  - Again with a regular GNU C backend
  - This time with a spaghetti stack (frames not copied)
  - But this time for parallel C/C++ for sales reasons
- See http://www.yl.is.s.u-tokyo.ac.jp/sthreads/
- This is a library that supports fine-grain multithreading in GCC/G++. still

Prospect: parallel languages are back ☺

- "parallel languages" used to be niche!

- but, people seems to start enjoying parallel platforms with the advent of
  - multi-socket multicore machines,
  - 8 way multicore/node × 1000 nodes are something you can buy from Amazon EC2 today

Prospect:
Super Lightweight Concurrency is back

- But, lightweight threads are available cheaply.

- "Super lightweight concurrency" is an old idea, but still a critical technique the PL community can contribute to, and

  it will be used extensively in near future.

Applications

- N-body simulation via Barnes-Hut algorithm
- CO-based Parser for Context-Free Grammar
- Linden’s “Second Life” / Online Virtual World

N-body simulation by Concurrent Objects

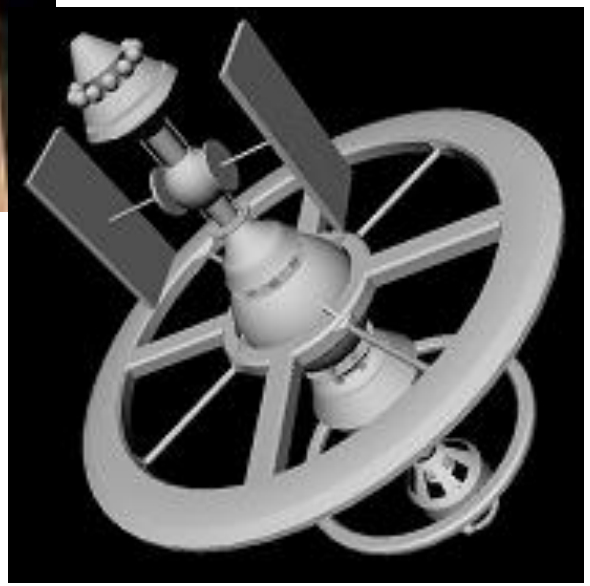
- ***concurrent objects represent*:**
  - **stars(masses)**
  - **center of gravity of stars**
- ***each concurrent object carrys:***
  - **xyz-position, velocity, weight**
- **employed Barnes-Hut method**
- **in 1995, computed with massively parallel machine (AP1000) of 512 SPARC nodes, StachThreads based implementation was used!!!**

Barnes-Hut Algorithm

- **Barnes-Hut algorithm performs an N-body simulation.**
- **Notable for having order O(n logn), compared to direct-sum algorithms which would be O(n2).**
- **The simulation volume is usually divided up into cubic cells via an octree,**
  - **so that only particles from nearby cells need to be treated individually, and**
  - **particles in distant cells can be treated as a single large particle centered at its center of gravity.**

Dynamics and Control of SpaceStation

- Rigid bodies and joints are represented as COs.
- COs caluculate torques and forces for stabilizing spacestation.



Mobile Concurrent Objects

Realization of Self-migration/mobile Concurrent Objects – JavaGo Language

H.Masuhara

- **JavaGo Language and its implementation that enables programmers to write concurrent objects moving around network nodes (1999)**

T.Sekiguchi

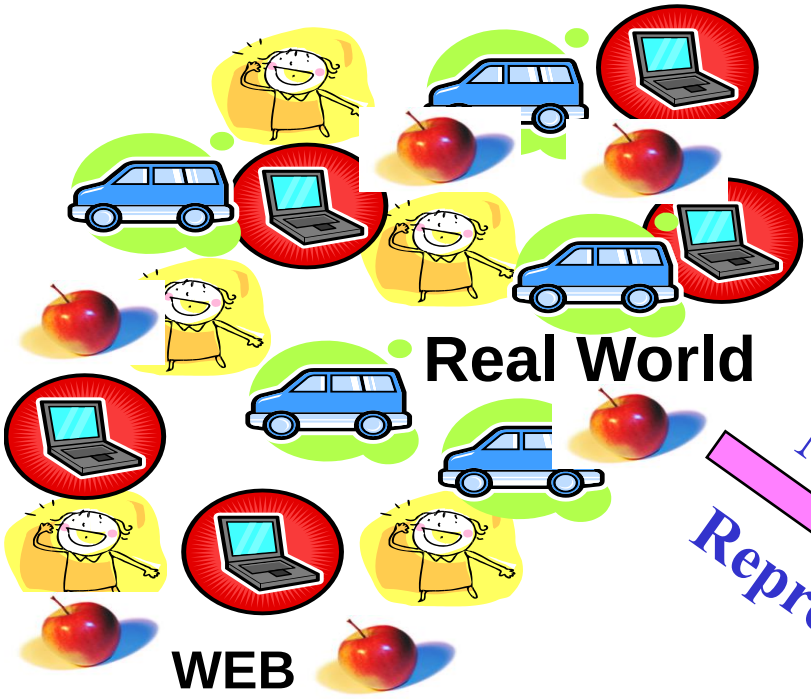
JavaGoX: transformation for transparent thread migration

- Java's support for mobile objects
  - dynamic class loading
  - "serialization" of object states
- JavaGoX enables efficient migration of ***running*** objects
  - by inserting code for saving/restoring execution stack into heap
  - implemented as a bytecode transformation system

cf. Sakamoto, Sekiguchi, Yonezawa: Bytecode Transformation for Portable Thread Migration in Java, in ASA/MA'00, 2000 for detail.

Massive Use of Concurrent Objects

Back to Original Motivation of COs

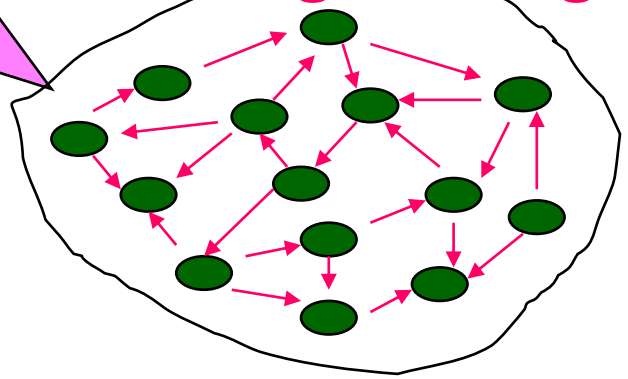


Modeling

Representing

Concurrent Objects &

Message Passing



Entities,people,
machines &
their interactios

Linden's *Second Life* …

is a natural outcome from the motivation of COs:

Concurrent Objects in Second Life

- Linden's online Virtual World that millions people participate in!

•Objects and avatars are represented and programmed as *concurrent objects!!*

Image from "Programming Second Life with the Linden Scripting Language" by Jeff Heaton (http://www.devx.com/opensource/Article/33905)

COs in Second Life

- Jim Purbrick, Mark Lentczner,

  "Second Life: The World's Biggest Programming Environment",

  Invited Talk at OOPSLA2007, said:

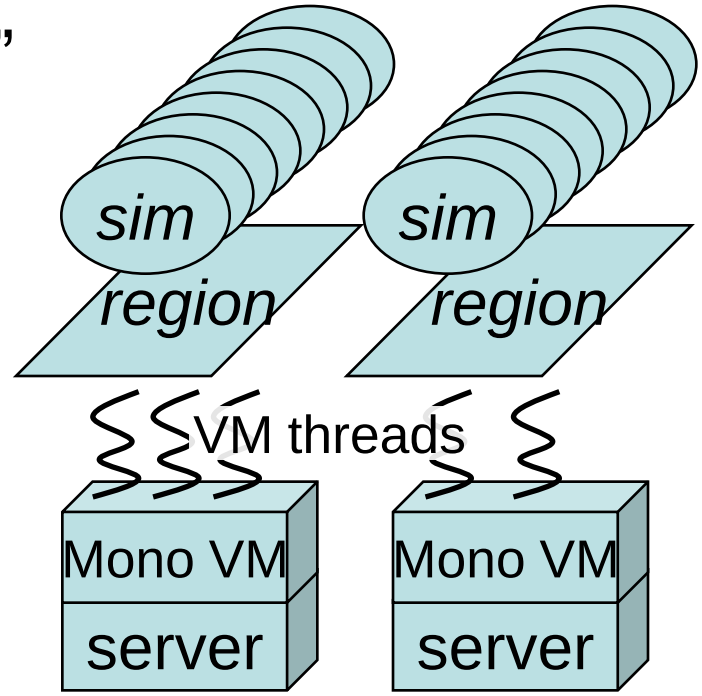
  - Objects and avatars cooperate and coordinate each other by exchanging messages.
  - each object or avatar is programmed to
    - **Have its own state,**
    - **Have its own method to respond to an incoming message,**
    - **Have different responses to different states, and**
    - **Have its own thread.**
  - About 2 millions of objects are programmed in Second Life and they are in action.

COness!

Second Life's new scripting engine on Mono*

They have a new implementation of Second Life!

- for accommodating many more "sims (simulated objects/Cos)"
  - a region constantly runs 1000s of scripts;
- for migration of sims between "regions"
  - even when they are running



***Mono: MS CLI compatible open source runtime**

*Purbrick (babbagelinden)'s blog on "Microthreading Mono", May 2006

Application of JavaGoX's transformation method to Second Life

- our JavaGoX [ASA/MA'00]
  - a bytecode transformation system

    that enables migration of ***running*** objects on JVM
- Second Life employs similar transformation for their new Mono-based script execution engine
  - for migrating objects between "regions"
    - **a region is managed by one server**

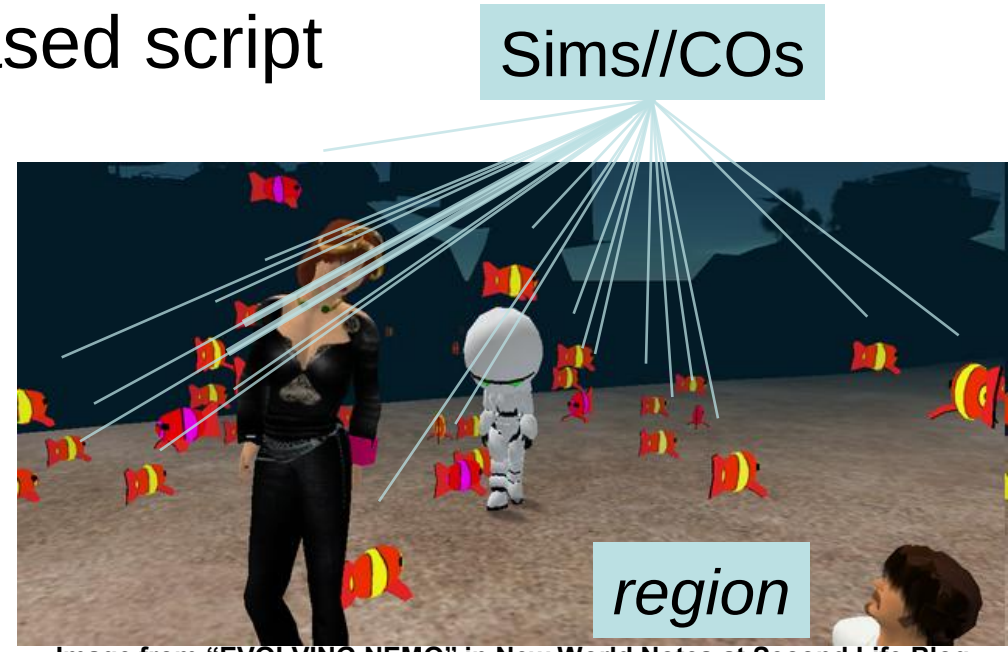


Image from "EVOLVING NEMO" in New World Notes at Second Life Blog (http://secondlife.blogs.com/nwn/2005/06/evolving_nemo.html)

Prospects

Why COs for Second Life

- The idea of concurrent objects has been adopted in Second Life because:
  - COs can *directly simulating* virtual world objects,
  - which enables
    easy modeling and
    easy/safe *concurrent* programming!

Why COs for Erlang and Revactor

- **Erlang:** *popular for distributed, fault-tolerant as well as WEB applications*
- **Revactor:** *actor/CO model implementation for Ruby, popular for web applications*
- **Both use**
  - asynchronous message passing communication not via shared variables,
  - super-light-weight thread with mailbox and send & receive
- **Why:**
  - No need for lock/release operations

  => **Easy/safe concurrent programming!!!**

Multi-Core Machines are Coming

- 2, 4, or 8 way multicore/node now available
- To maximally exploit such machine power, need to manage super-light-weight threads with no shared memory communication

  with tiny cost!!
- Now this is possible!!

We are winning…

"Concurrent Object" enjoys:

- natural and powerful modeling,
- easy and safe concurrency/thread managing,
- super-light weight thread implementation technology (such as *StackThreads*) is available,
- multi-core hardware architectures more popular.

We will be able to do much finer, more powerful modeling/simulation/programming of {real and virtual worlds} such as physical, social, organizational,…,phenomena!!

Thank you for your attention!!