並列オブジェクト

研究を省って、今後に向けて

米澤　明憲

東京大学　情報理工学系研究科
コンピュータサイエンス専攻　＆
東京大学情報基盤センター

講演内容

- 着想を得た頃のこと
- プログラミング言語研究の正四面体
  - 言語設計
  - 自己反映計算
  - 線形論理による意味論
  - 超並列コンピュータ上での言語実装
  - 自発的移動オブジェクト- *JavaGo*
  - 応用例:
    - N-体問題, 宇宙ステーションの計算…
- 並列オブジェクトの大規模応用
  - リンデン社のセカンドライフ-
- 今後の研究

MIT　B.Liskovのセミナー (1974)

ダールさんの本

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

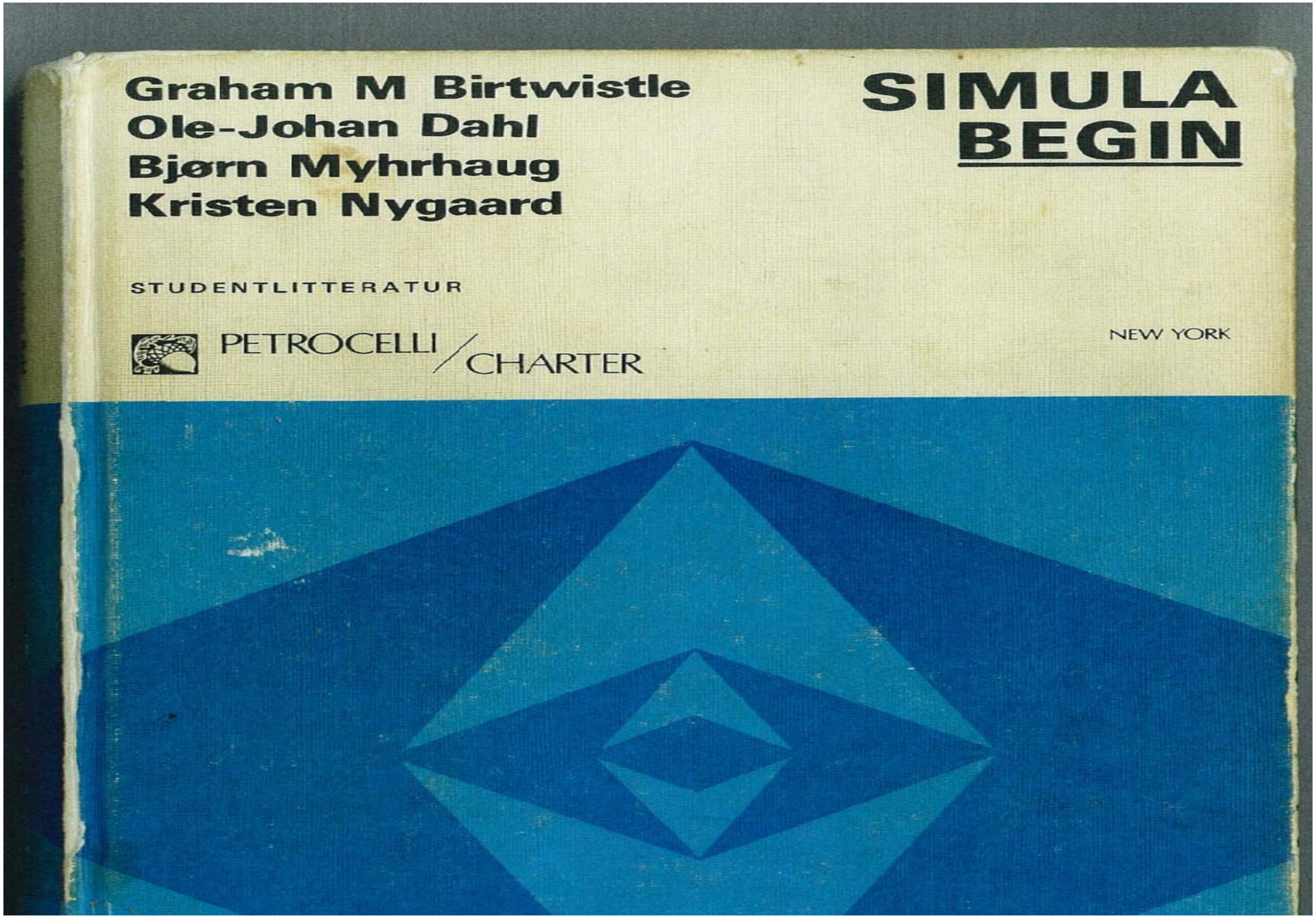
C. A. R. HOARE
Department of Computer Science,
The Queen's University of Belfast,
Belfast, Northern Ireland

1972
ACADEMIC PRESS
LONDON AND NEW YORK

A Subsidiary of Harcourt Brace Jovanovich, Publishers

C.Hewittの示唆 (1976)

ニゴールさんの本



Simula67の後，そして70年代前半

- 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

いかに、プログラム表現や知識の表現形式を
構造化・モジュール化にするかの探求

モデル化とプログラミングの両方を！

70年代の初め東京で、プログラミング言語と定理証明を勉強した。
その後, 興味を持ち出したのは:

1. 「世界」のモデルをつくり、それに従って「世界」をコンピュータの上でシミュレートする。

2. より強力なプログラム方法論をつくること。

プログラミング研究の目的とオブジェクト指向

- 実行性能を維持しながら、ソフトウエアシステムの**記述の複雑さを減少**させること
- ソフトウエアシステムやその構築をより単純化し管理しやすくできる

構築するソフトウエアシステムをより強力なものにできる。

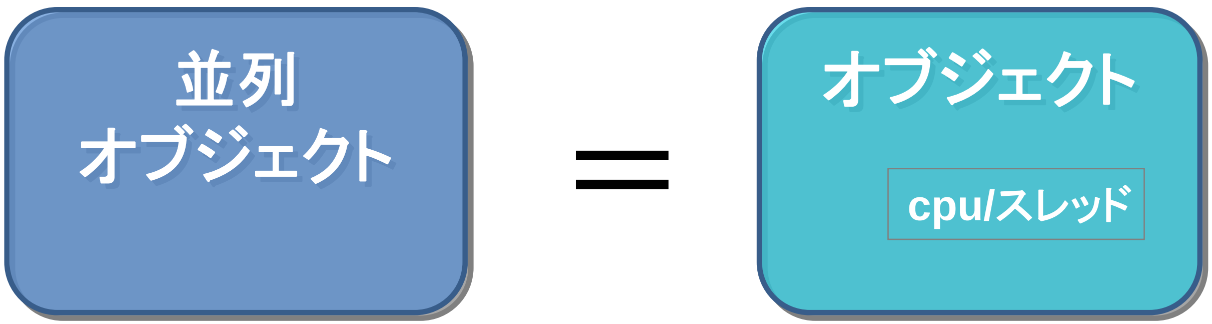
オブジェクト指向法が生まれた!!

1974年ごろそのアイデアを思いついた

モデル化とプログラミングの目指して,...

- 並列オブジェクト

= カプセルに閉じ込める(状態をもつオブジェクト + スレッド)

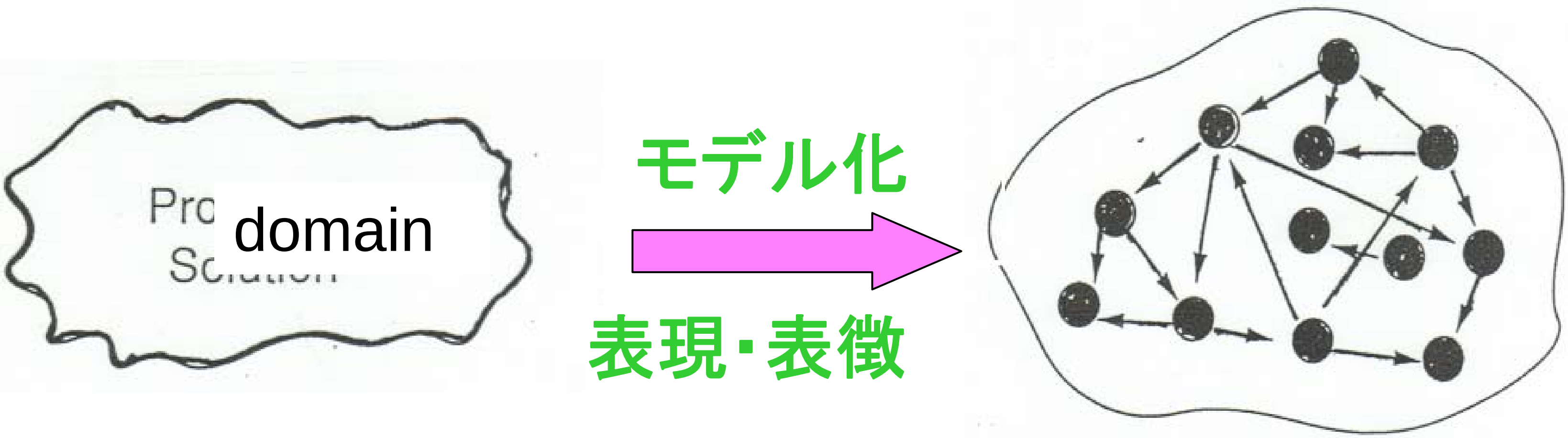


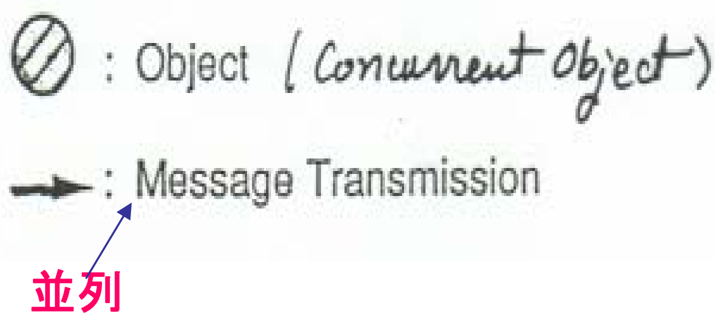
- 並列オブジェクト同士の非同期的メッセージ送信

- 少し違ったアプローチ(Actor)

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

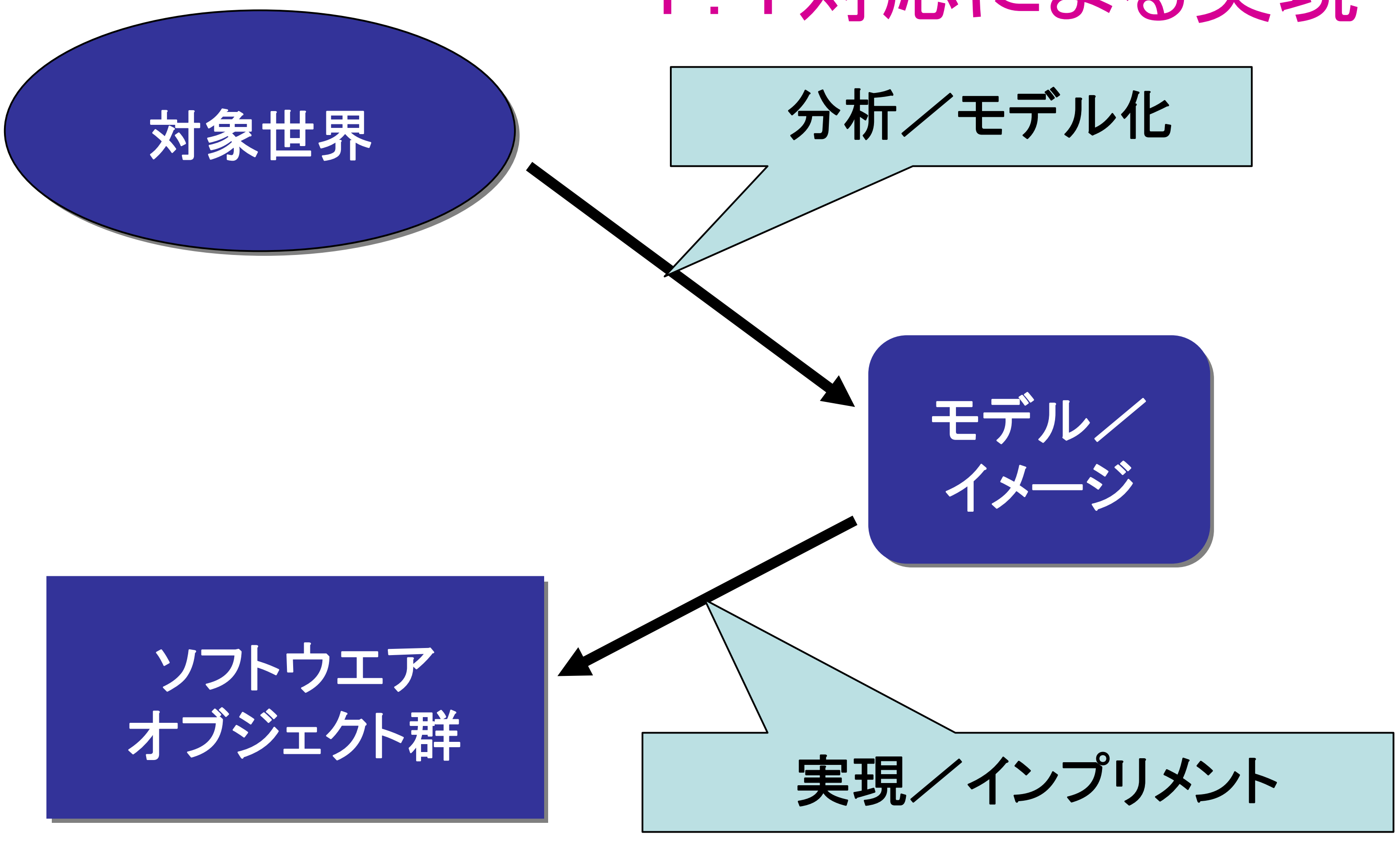
並列オブジェクトによる「世界」のモデル化





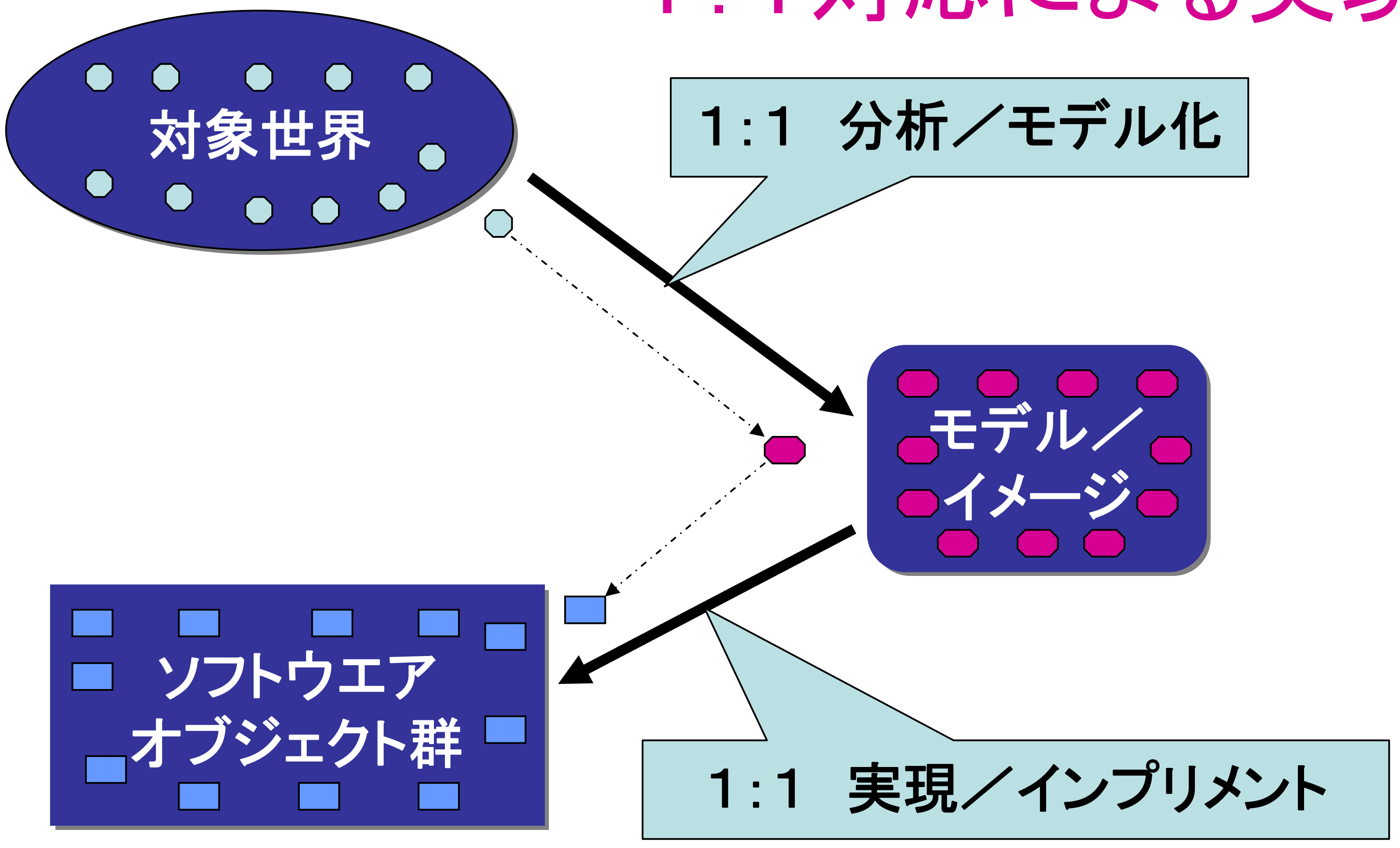
自然／直接的なモデル化と実現

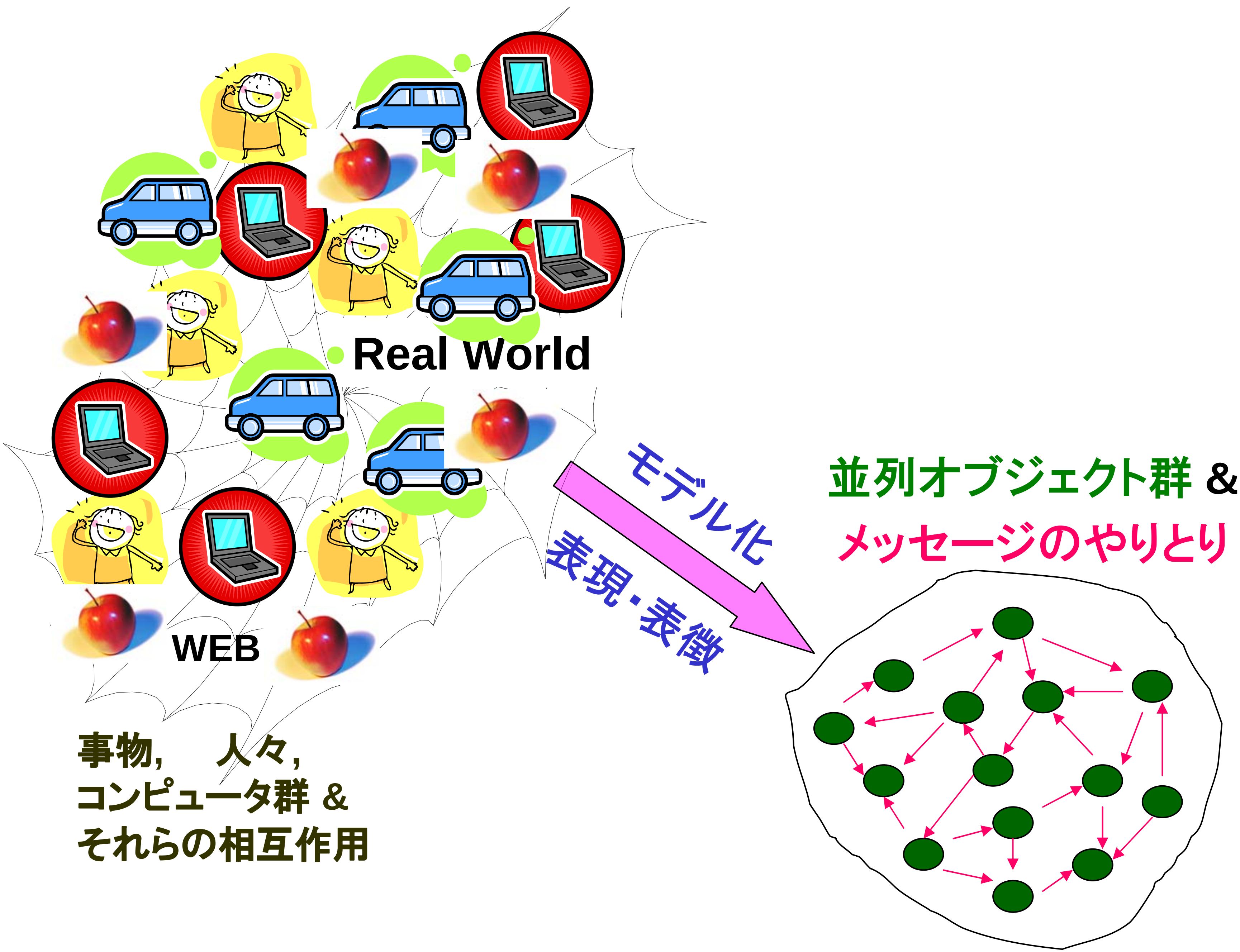
1：1対応による実現



自然／直接的なモデル化と実現

1：1対応による実現

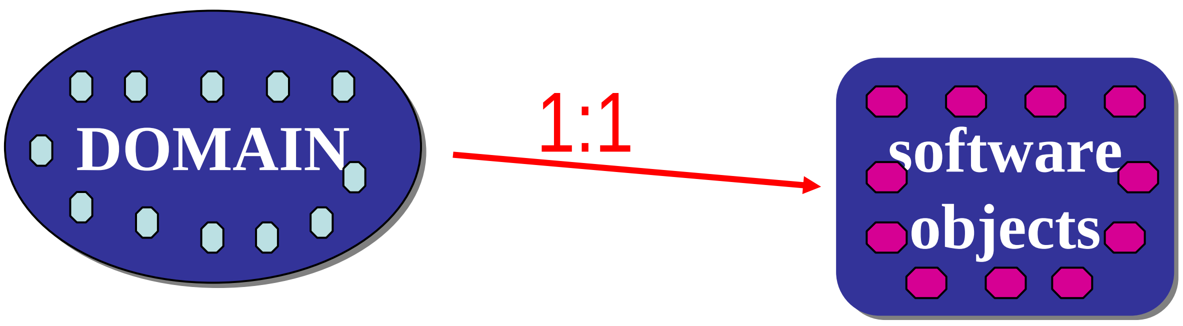



Real World
WEB
事物,　人々,
コンピュータ群 &
それらの相互作用
モデル化
表現・表象
並列オブジェクト群 &
メッセージのやりとり

「世界」の自然なモデル化

- 自然なモデル化 が 複雑化を防ぐ・減らす
- 「自然さ」とは 直接さのこと、「直接さ」とは？
  - 問題領域(domain)での登場物と

    ソフトウエアモジュール群との**1:1 対応**

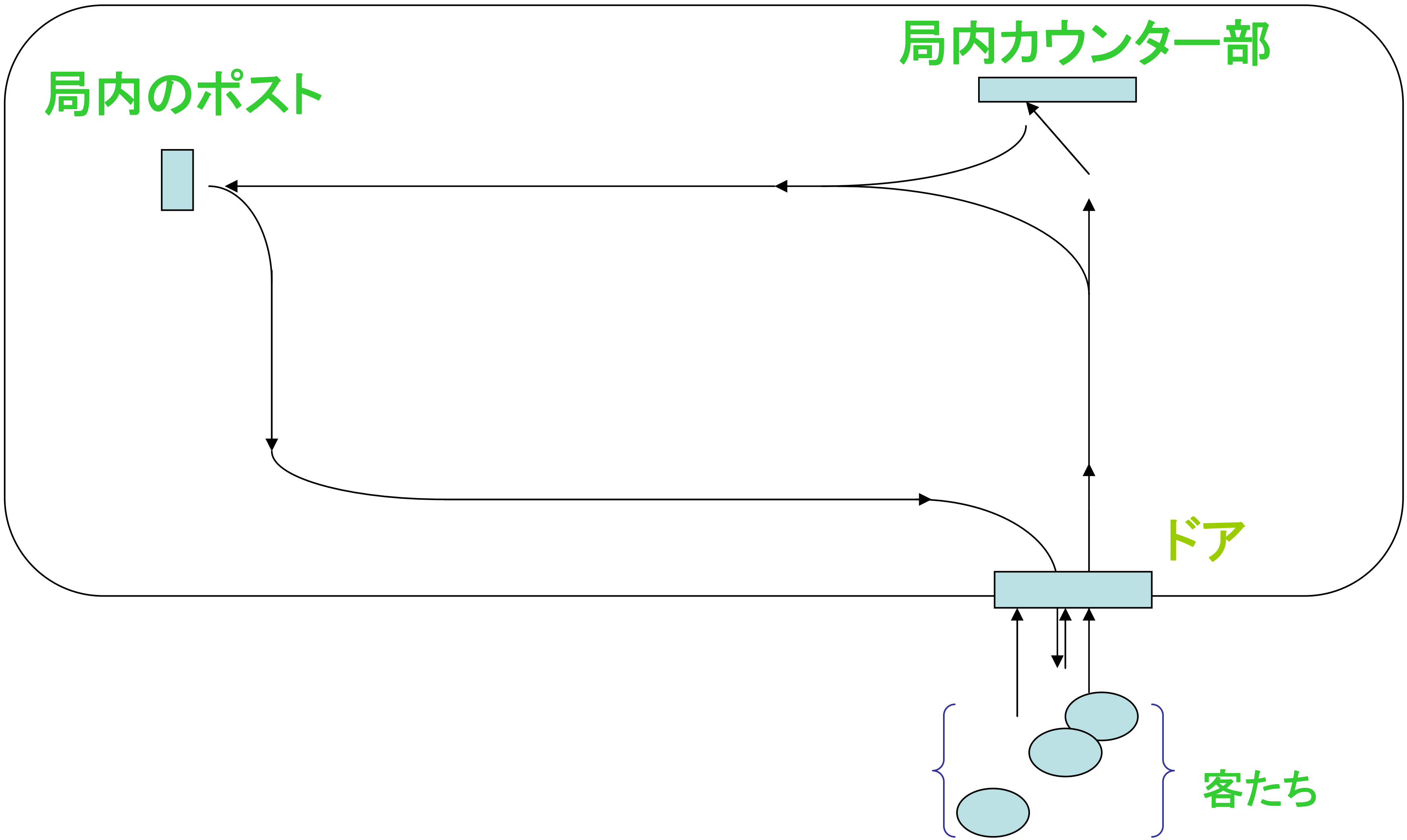


- Ole Madsen said:
  - Objects and Classes are **well-suited** for modeling **physical entities** and associated concepts
  - **“Concurrency” is *MUST* for modeling**

自立移動オブジェクト

例:

小さな郵便局をモデル化('77)



小さな郵便局を並列オブジェクトでモデル化する

- 郵便局の建物 ⇔ ドア
  - ➔ ドアを表現する並列オブジェクト
- 局員のいるカウンター
  - ➔ カウンターを表現する並列オブジェクト
- 郵便ポスト
  - ➔ ポストを表現する並列オブジェクト
- お客たち
  - ➔ お客たち並列オブジェクト群

    （注意、メッセージで表現しないこと）

お客の移動をモデル化する

- ２つのやり方:

1. 客オブジェクトをメッセージに入れて
   メッセージを送信する
2. 客オブジェクト自身が自分で移動する。

オブジェクト (あるいはそのコードが) 移民する!!

環境・アンビアントから教えられる…

- 地元に住んでいない客は
  郵便局内の地図を知らない。

客オブジェクトは

カウンターオブジェクトの存在や位置を知らない

客オブジェクトはカウンターオブジェクトの位置や名前をドアオブジェクトから教えられる。

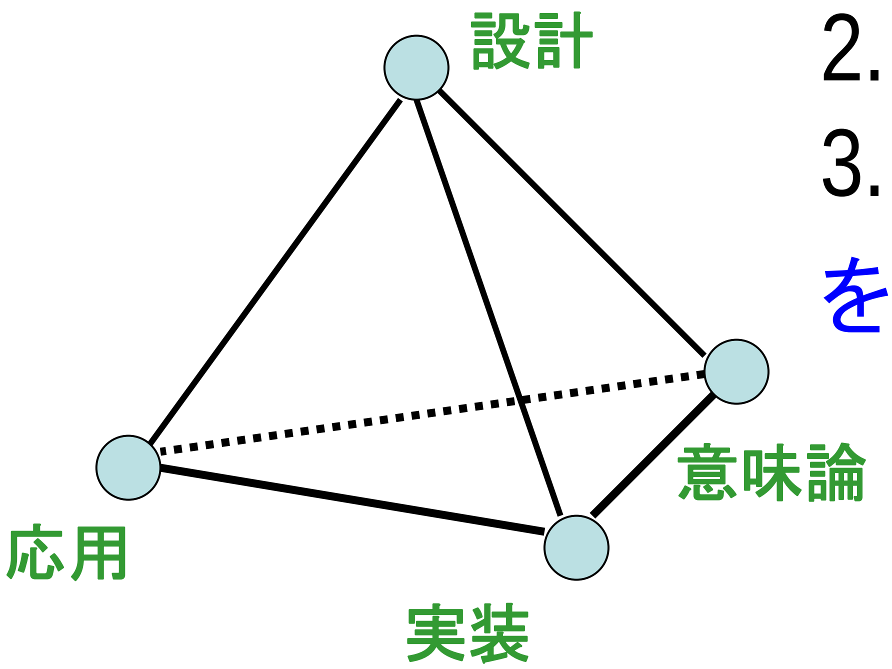
アンビアント

言語研究の正四面体

プログラミング言語を設計したなら、

1. 処理系の実装
2. 数学的意味論
3. 応用プログラム

を、きちんと研究すべし！



1984年以来の私の研究

- 言語設計

  ← *ABCL言語シリーズ* (JP Briot, Shibayama) 1984-

  ← Inheritance Anomaly (S.Matsuoka, JP Briot) 1985-1989

  ← **自己反映計算** (T.Watanabe) 1988
- 意味論

  ← **線形論理を基礎に** (N.Kobayashi) 1991-
- 超並列コンピュータ上での処理系実装

  ← StackThread**方式** (K.Taura) 1993-
- 移動オブジェクトとその実現

  ← JavaGo (T.Sekiguchi, H.Masuhara) 1999
- 応用プログラミング

  ← N-**体問題**, **宇宙ステーションの動力学**, CFG**解析**… 1997

共同研究者たち

柴山悦哉

J-P. Briot

松岡聡

小林直樹

田浦健次朗

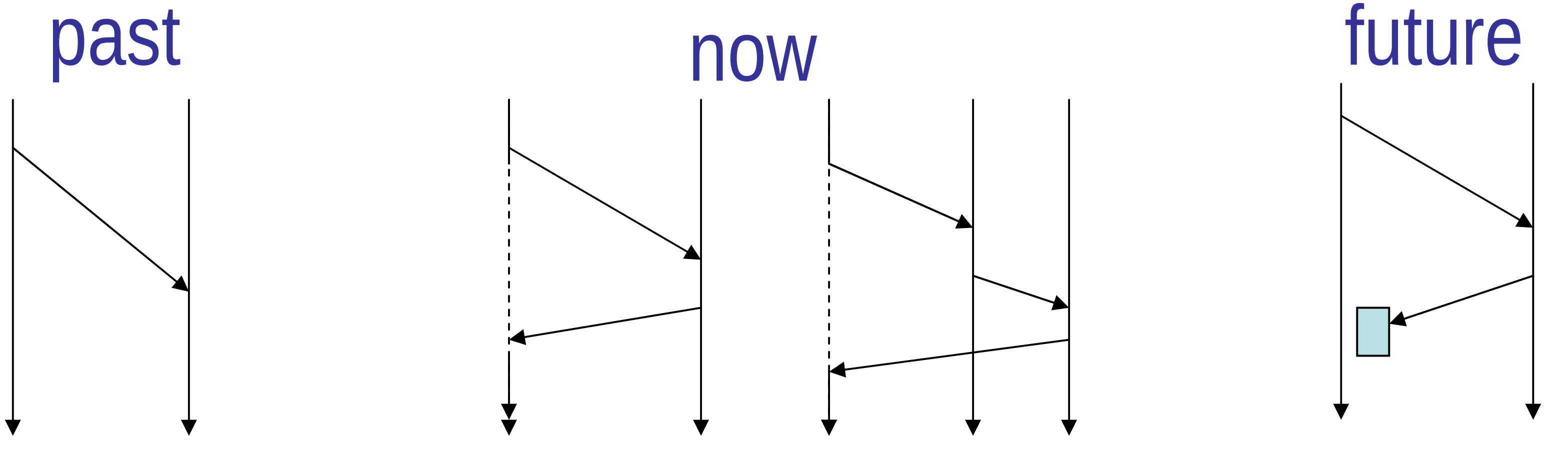
増原英彦

渡部卓雄

関口龍郎

ABCL/1言語でのメッセージのやりとり

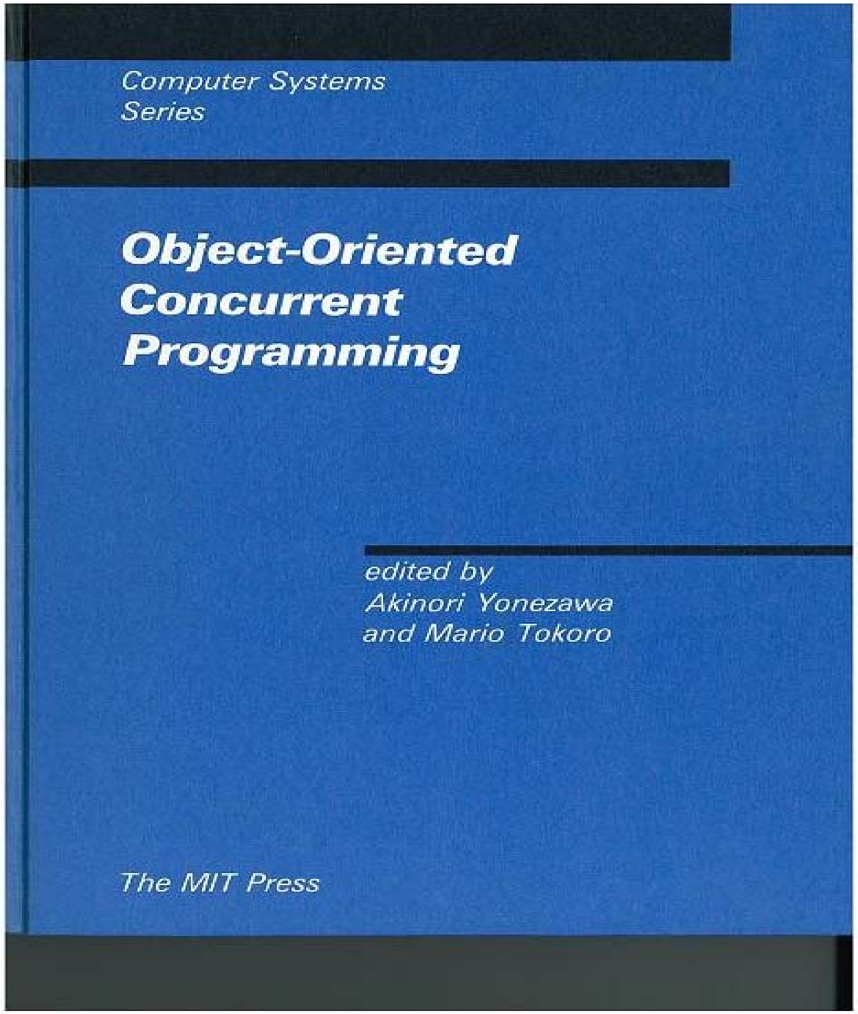
- メッセージのやりとりは全て非同期式
  - より自然なモデル化とより沢山の並列性
- メッセージ送信の 3つのタイプ：
  - 送りっぱなし (past)
  - 送って返事が来るまで待つ (now)
  - 「未来」付き送りっぱなし (future)

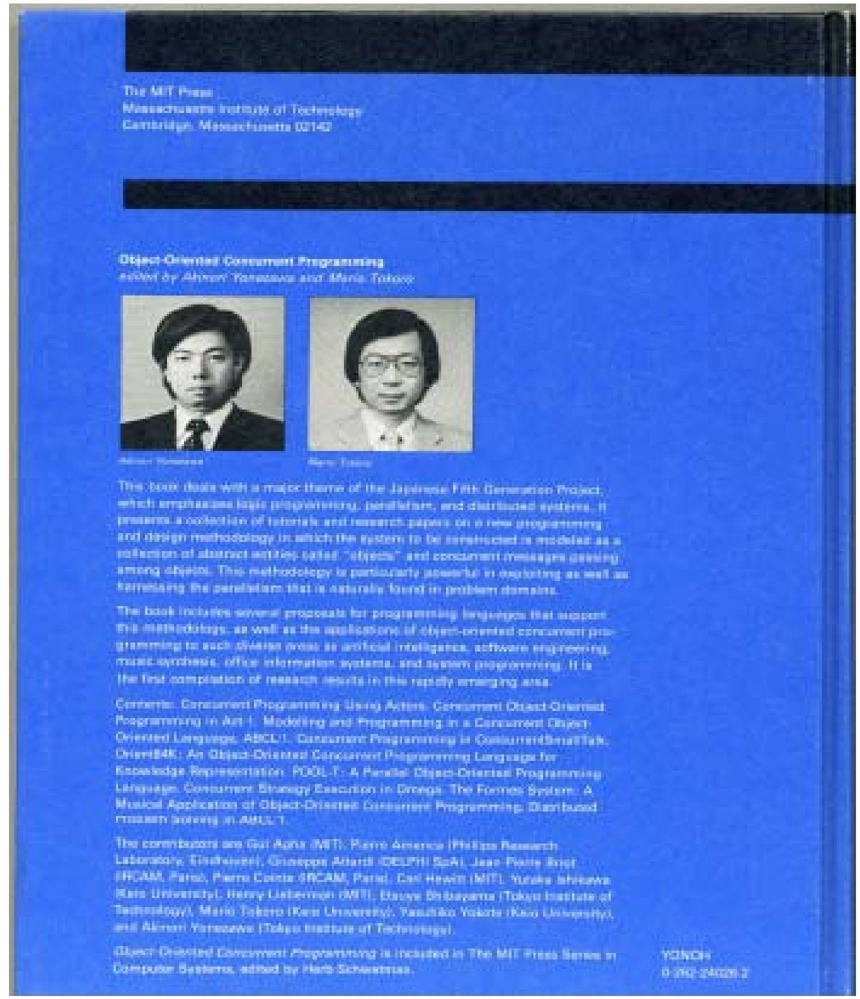


我々の最初の言語 ABCL/1 (1984)

- 世界で初の並列オブジェクト言語
- それぞれのオブジェクトは1つのスレッドを持つ
- 任意の時点で, 並列オブジェクトは
  次の3つのモードの 1つにある:
  (1) *休眠*, (2) *活性*, (3) *待機*
- 継承機構を持たない

Book in 1987


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



Contents

H.Lieberman

ABCL/1の
処理系実装と応用プログラム

- LISPベースの実装（SUN ワークステーション上）
- マニュアルとプログラミングガイドをつくり
  OOPSLA'86で配布
- Lispマシン上での よりしっかりした処理系の実装1987.
- 文脈自由文法の並列オブジェクトベースの構文解析器
  - **250以上の非終端記号をもつ英文法（1987）.**
  - **ブタペストでの計算言語学会1988に発表、反響多し (Computational Linguistic Conference in Budapest, 1988)**
  - **その後、ヨーロッパから出版さた並列構文解析に関する本にヨーロッパ研究者の関連論文が多く収録された。**

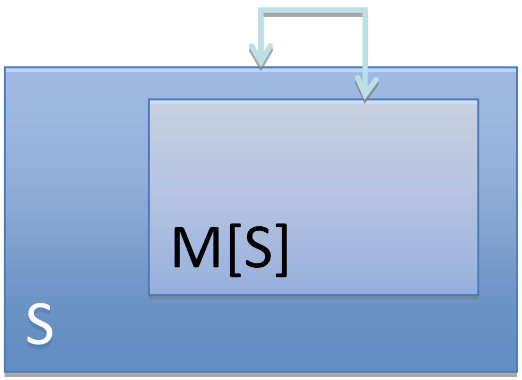
並列オブジェクトでの自己反映計算

- B. Smith, 3-Lisp(1982)に触発された
- P.Maes と L.Steels, 3-KRS(1984）に触発された

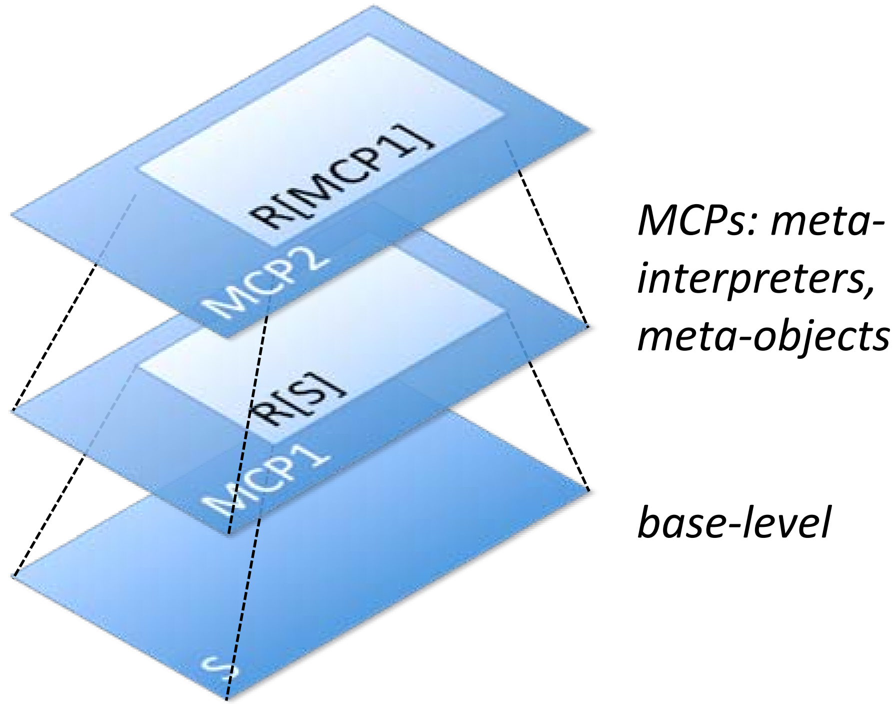
  – With 渡部卓雄(東工大）

自己反映計算とは
Computational Reflection

自分自身についての計算・情報処理: 内省と自己変更



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

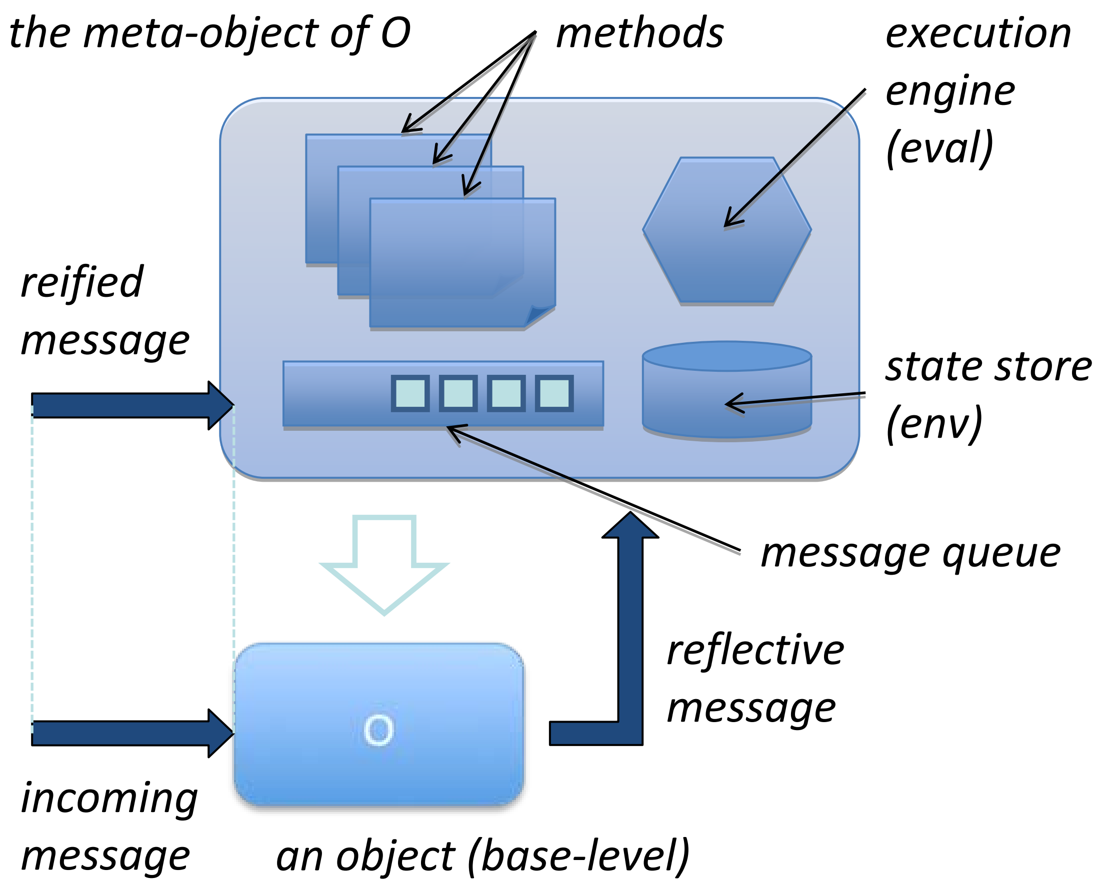
並列オブジェクトの解釈器を記述してみる

1つの並列オブジェクトA　–> 1つのメタオブジェクト(Aのモデル)

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

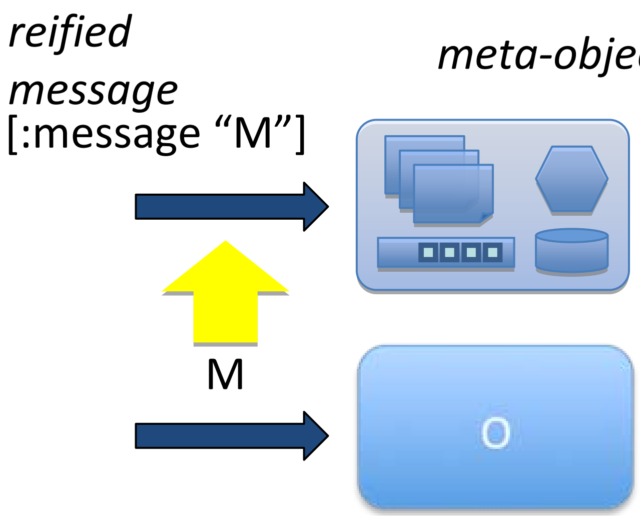
メタオブジェクトの動き (1)

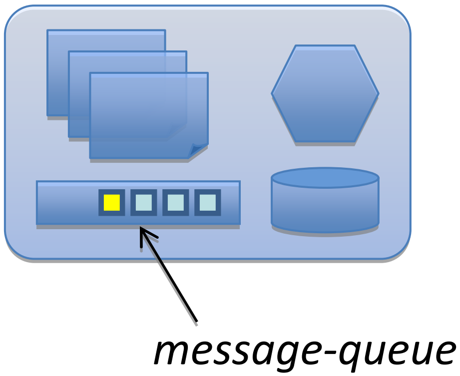
並列オブジェクトの超循環解釈器
Metacircular Interpretation of Concurrent Objects

(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]
```



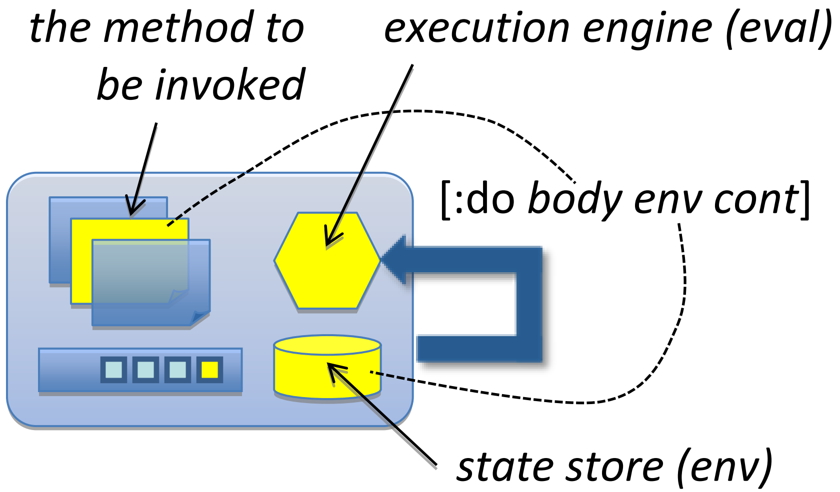


メタオブジェクトの動き (2)

並列オブジェクトの超循環解釈器

(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.

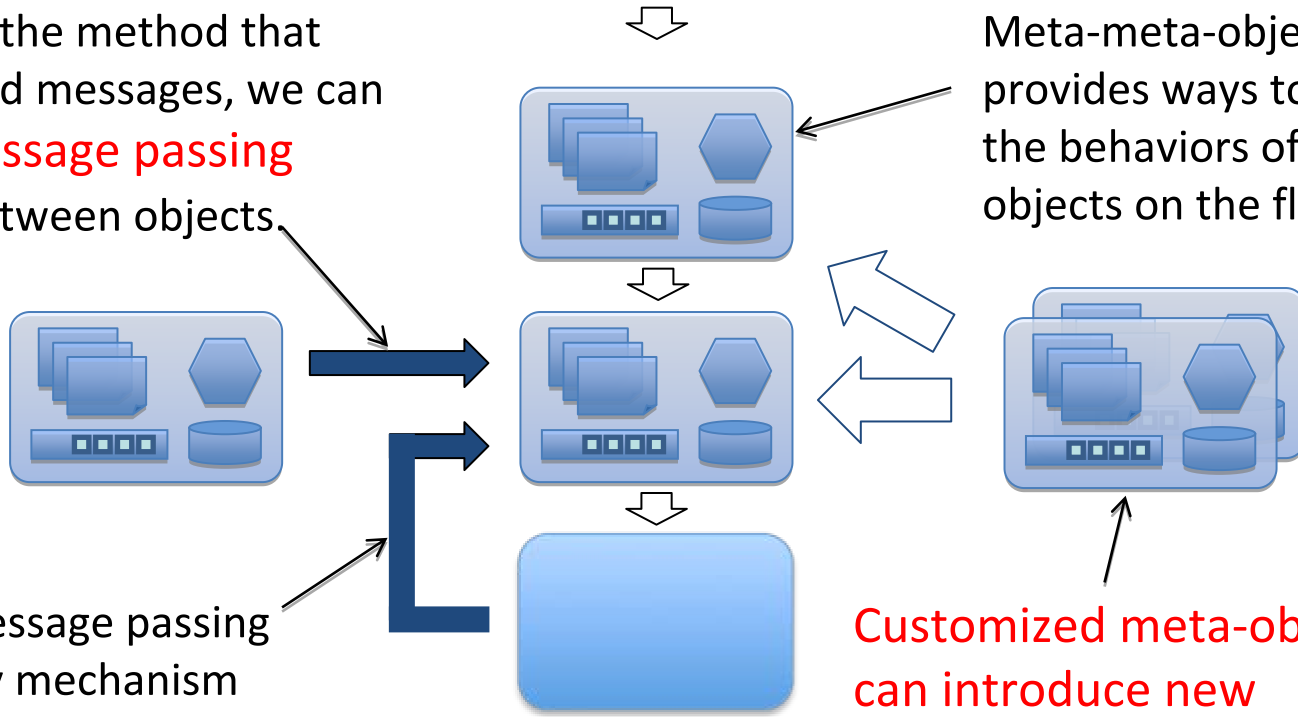
(5) The meta-object repeats the above actions while the queue has outstanding messages. When the queue gets empty, the object becomes dormant.

メタオブジェクトの使い道

自己反映計算で並列オブジェクトが持つメッセージ処理部やそのほかの部品を変更できる部品!!

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.

ABCL/Rを用いた応用

- 動的にメッソドを獲得できる
  - レヴェル間のメッセージのやりとりの例
- 着信・送信するメッセージの監視器の動的着脱
  - Meta-meta-objects are used to add/remove monitoring in/out-messages in meta-objects.
- メッセージ送受信機能へのタイムワープ機構のモジュラーな導入
  - Customized meta-objects provide an encapsulated implementation of the algorithm.
    - Timewarp Algorithm: an optimistic algorithm for distributed discrete event simulations (by D. Jefferson, 1985)

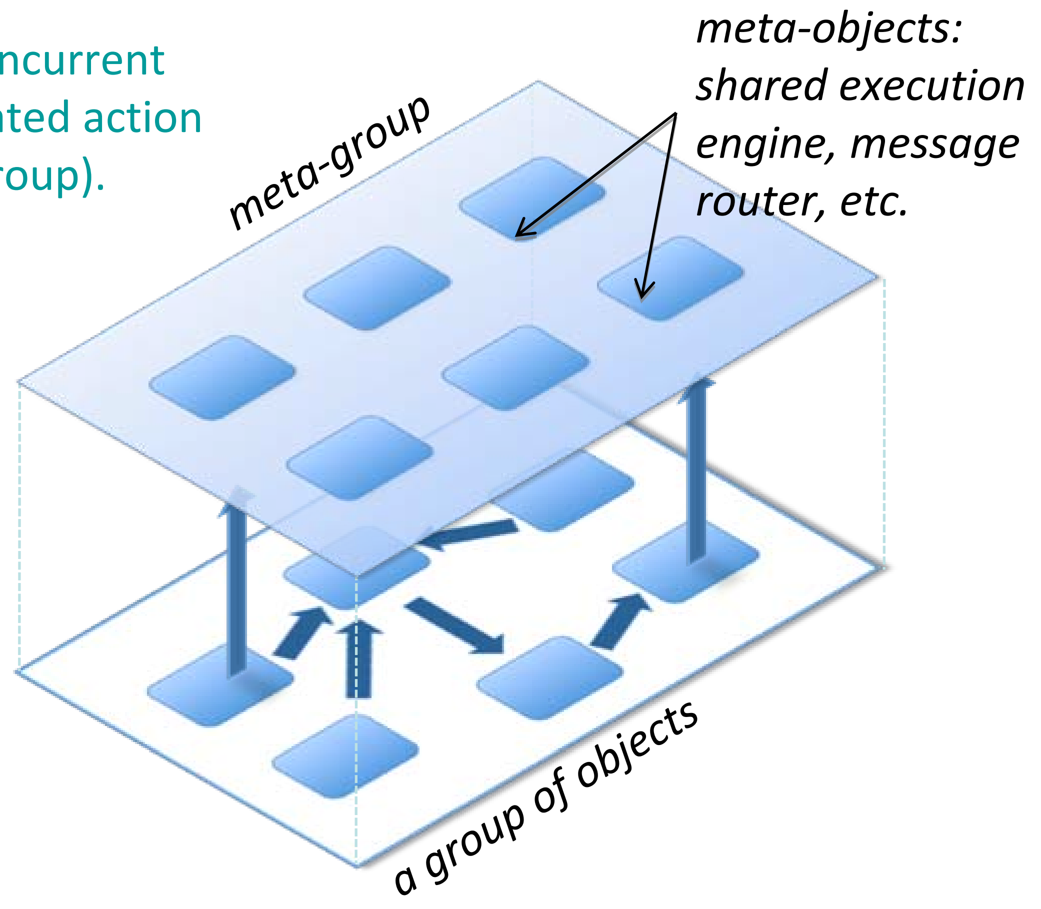
オブジェクト群の共同自己反映計算

オブジェクト群全体を処理する一括したメタレヴェル

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.

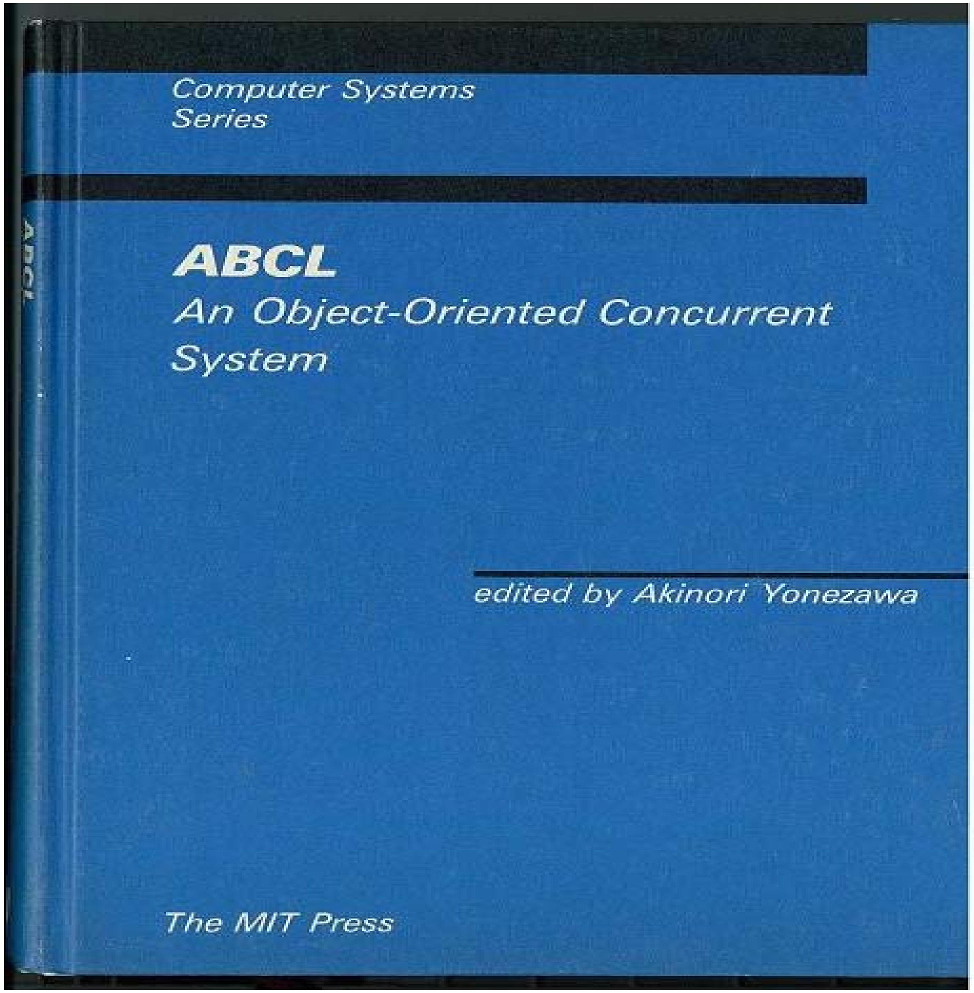


応用例:
オブジェクトの動的移民,
適応的スケジューラ, etc.

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

Book in 1990

- 1989年までの我々の研究論文や言語マニュアルなどを収録した本.(スタンフォード大の書籍部で平積み)
- 以下の研究は収録されていない:
  1) StackThreads
  2) JavaGo言語の処理系実装と数学的意味論



線形論理を基礎とする意味論

- 並列オブジェクトに基づくプログラム言語の形式的・数学的意味論が欲しかった！

- R. Milnerのπ-計算を用いて構築するのも1つのやり方ではあったが。。。
  - π計算の帝国に紛れ込むのはちょっと心残り
  - 私はGentzenスタイルの sequent形式に慣れていた！！

$\Rightarrow$

Girardの線形論理を使うことにした！

並列オブジェクト指向言語の意味論

小林直樹（東北大）

- 1991にプロジェクトを開始
- 目標:
  - 以下の目的のために並列オブジェクト指向言語の数学的基盤を構築する:
    - 言語設計の指針得る（型システムの構築を含む）
    - コンパイラがするコード最適化の正当化
    - プログラム正当性の証明
    - *研究上の尊敬（research prestige）を得る*

線形論理とは？

- 資源の消費を意識する 論理体系[Girard 87]

  A –o B *線形含意*

  Aを**消費する**ことによって**B**が導出される

  （1度しかAであることを推論に用いない）

  A ⊗ B テンソル積

  A と B が同時に成立する

  A & B

  A と B が成立するが, but 同時には成立しない
  (AかBのどちらかを選択しなければならない)

  !A

  Aは何度でも成立する。何度も推論の使える。

線形論理のエッセンス

- 例
  - A: 100円持っている
  - B: （100円の）コーラが手に入る
  - C: （100円）のチョコレートが手に入る

A –o B　常に真

A –o C　常に真

A –o B⊗C　いつも真ではない

(100円でチョコレートとコーラは同時には買えない）

A –o B&C　常に真

(100円あれば、

チョコレートかコーラかどちらかは必ず買える)

並列オブジェクトを線形論理式として表現する

- *m -o A*
  - メッセージ mを*受理/消費して*、その後 Aのように振舞うオブジェクト
- *m ⊗ A*
  - メッセージmを*送信し*, その後 Aのように振舞うオブジェクト
- *論理推論を計算とみなす(deduction as computation)*

(c.f. logic programming)

$$\underbrace{(m \otimes A)}_{\text{送信者}} \otimes \underbrace{(m \ \text{-o}\ B)}_{\text{受信者}} \ \ \text{-o}\ \ A \otimes B$$

並列オブジェクトを線形論理の論理式として表現する

$$
\begin{array}{l}
!\forall n,inc,read. \\
\quad (counter\ (n, inc, read) \multimap \\
\quad\quad \forall reply.(inc\ (reply) \multimap \\
\quad\quad\quad\quad (counter\ (n+1, inc, read) \otimes reply())) \\
\quad\quad\quad \& \\
\quad\quad \forall reply.(read\ (reply) \multimap \\
\quad\quad\quad\quad (counter\ (n, inc, read) \otimes reply(n))))
\end{array}
$$

並列オブジェクトの型理論
[OOPSLA 1994]

- 論理式型をプロセス型として
  - O

    論理式の型

    ≈ オブジェクトとメッセージの型
  - int → O

    整数に関する述語の型

    ≈ 整数を授受するチャネルの型

    e.g. ∀x:int.(c(x) –o d(x+1))
  - (int→ O) → O

    整数に関する述語につい点の述語の型

    ≈ 整数を授受するチャネルを授受するチャネルの型

    e.g. ∀x:int → O.(c(x) –o x(1))

主要論文

[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)

超並列マシン上での
言語処理系の実装

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

ABCL on 富士通AP1000超並列マシン上のABCL言語の実装 (1992-)

- 超並列マシン上で並列オブジェクト指向言語ABCL/f 言語の処理系の一連の開発
- ハイパフォーマンス計算を狙って開発開始

超分散メモリ型並列マシンを
ターゲットとしたハイパフォー
マンス計算用の並列言語
処理系としては、初期の数
少ない開発

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

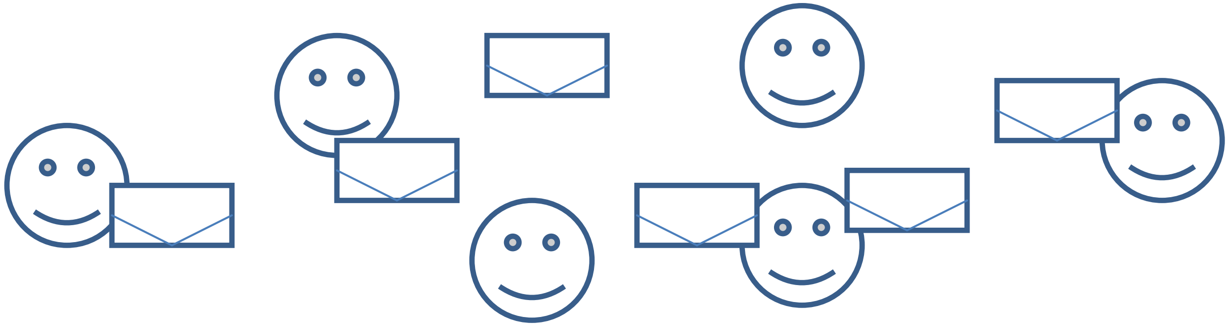
AP1000 with
512 nodes

1992年には。。。

- プロセッサアーキテクチャには種々のものがあった
  - **データフローマシン**: *T, EM4, J-Machine
  - **超並列マシン**:　AP1000, CM5, -- ccNUMA: DASH
- 種々のプログラミング言語
  - **オブジェクト指向**:　ABCL, Concert, …
  - **関数型**:　Multilisp, Id, Sisal
  - **論理型**:　KL1
- 我々はもちろんオブジェクト指向型で、future方式をベースにするABCL/f 言語は超並列マシンAP1000に実装した!!

我々がまず探った方式は。。。

- 並列オブジェクトの実行モデルから:
  - *“並列オブジェクトは自分自身のスレッドを持ち、オブジェクトの間でメッセージのやりとりをする、*



- これは以下のように考えられる:
- *並列オブジェクト = データ + 1つの実行スレッド*
- しかしこの考えでは旨く行かなかった、 なぜならスレッドがとてつもなく資源を使う

通常のライブラリとして提供されるスレッド

様々なアイデアをテストした。。。

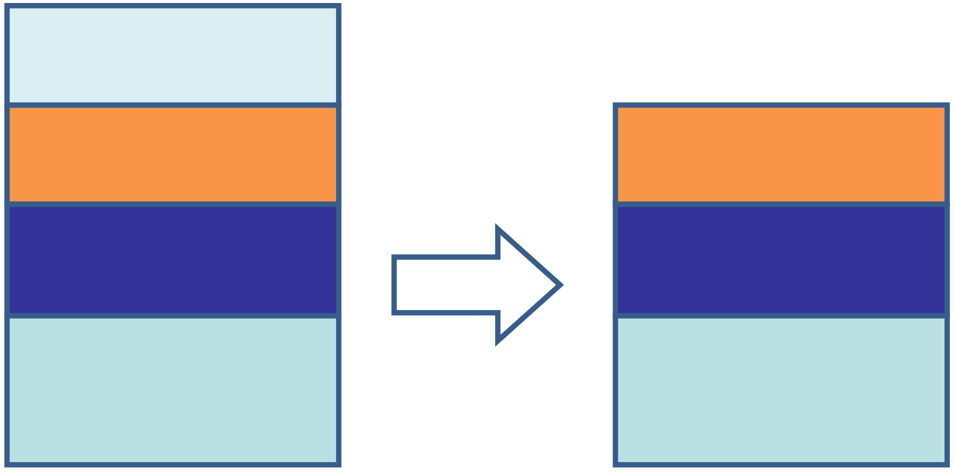
- 試み1: “スレッドプール”と呼ばれる方式
  - そのままスレッドを使うよりはましだが,
- 試み 2:　スレッドを非同期メソッドに貼り付る方式
  - 何百万もの非同期メソッド呼び出しに対応してスレッドが作られ大変。。。
- 試み3:　“スタックスレッド”方式というものを考案
  - *1つのスタックだけを使って全てのスレッドを投機的に実行する。*

“StackThreads方式”

- 唯一のスタックで全てのスレッドを実行
- では、スレッド切り替えをどうするのか?
  - 単純!、スタック内ポインタの操作だけで行い、かつスタックフレームの削除はスタックトップから行う
- 大変低コスト かつ 高速のスレッドが得られた

=>

この結果、莫大な数の細粒度スレッドが使用可能となった。



スタックスレッズ (続き)

- GNU C コンパイラを細工してこのスタック方式を実現し、バックエンドとして使う方式 を考案(PLDI '97)
- 共有メモリマルチプロセッサマシンにスタックスレッズを実装し、プロセッサ切り替えにはワークスティール方式を使用するアプローチを考案 (PPoPP '99)
  - 前と同様に、GNU C コンパイラをバックエンドとし利用
  - ここではフレームをコピーしないスパゲッティスタック
  - parallel C/C++ で使えるようにした

See http://www.yl.is.s.u-tokyo.ac.jp/sthreads/
これは細粒度マルチスレッドライブラリで
GCC/G++で使える。今でも ダウンロードで使用可能

展望:
並列言語が戻ってきた☺

- “並列言語” は　隙間研究(nichie)であった!
- しかし今は, 並列プラットフォームを使うのが流行になりはじめる印象がある、なぜ
  - マルチソケット、マルチコアマシンの時代に到来
  - 8 ウェイマルチコアを1ノードとして、1000ノードが　Amazon EC2 で使える昨今である。

展望:
超軽量並列が再び脚光を

- 軽量スレッドは安く使える.

- “超軽量並列が” 古くからのイアデアであるが、

プログラミング言語コミュニティが貢献できる強力なテクノロジーであるので

このアイデアは

今後広く使われると予想される！！

並列オブジェクトの応用例

- Barnes-Hut アルゴリズムを用いた
  多体問題シミュレーション
- 並列オブジェクトベースの文脈自由文法構文解析器
- Linden社の“セカンドライフ” /オンライン仮想世界

N-body simulation by Concurrent Objects

- *並列オブジェクトに:*

  - 星(質量)　や　星群の重心

  を表現させる！

- *各並列オブジェクトがその状態で:*

  - xyz-位置, 速度, 質量

- Barnes-Hut法を使用

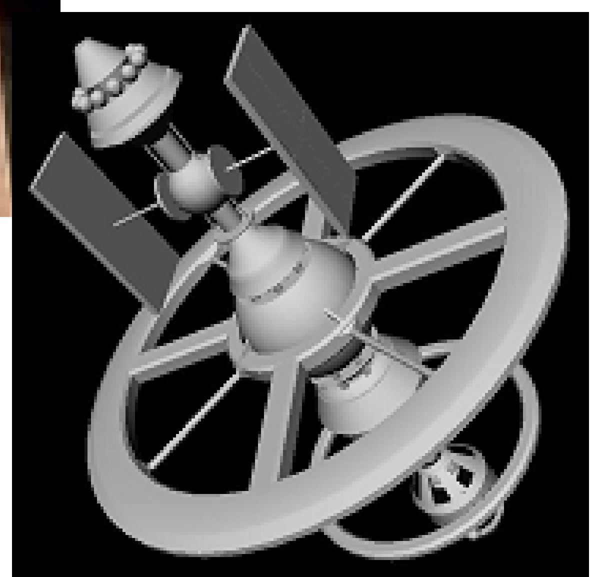
•1995年に, StachThreadsベースのコンパイララを用いて512 SPARC nodesのAP1000で実行

Barnes-Hutアルゴリズム

- **Barnes-Hut** アルゴリズム <u>N-体問題の良い解法</u>
- N個の質量にたいしてO(n logn)の手間, 一方総当り法ではO(n2).
- 対象質量が分散する空間を <u>octree</u>のノードが立方体となる形の再帰的に分解し、各リーフには1質量しかないようのする。
  - 近距離どうしの質量間の重力は正確の計算し、他方
  - 遠距離にある質量群の影響は質量群がその重心位置にあると近似して、作用する重力を計算する。

宇宙ステーションの動力学と制御

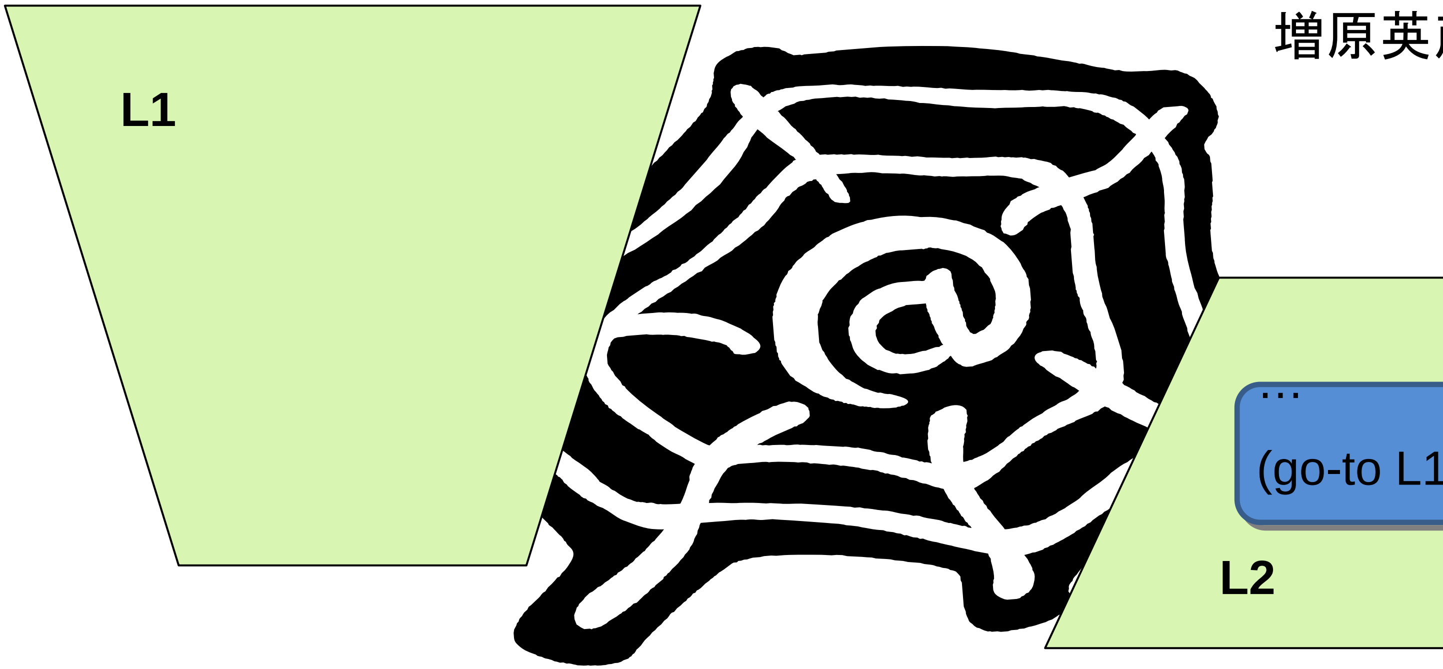
- 剛体と結合を並列オブジェクトで表現する.
- 並列オブジェクト群がトルクや力を計算して宇宙ステーションの動きを計算あるいは安定させる力のかけ方を計算する.



自立移動オブジェクト

自立的に移動可能な並列オブジェクトのプログラミング– JavaGo言語

増原英彦



- ネットワーク上を自由の自立的に
移動可能な並列オブジェクトをサポートする
JavaGo 言語とその実装 (1999)

関口龍郎

JavaGoX: 透過な移民を可能にするプログラム変換に基づく言語

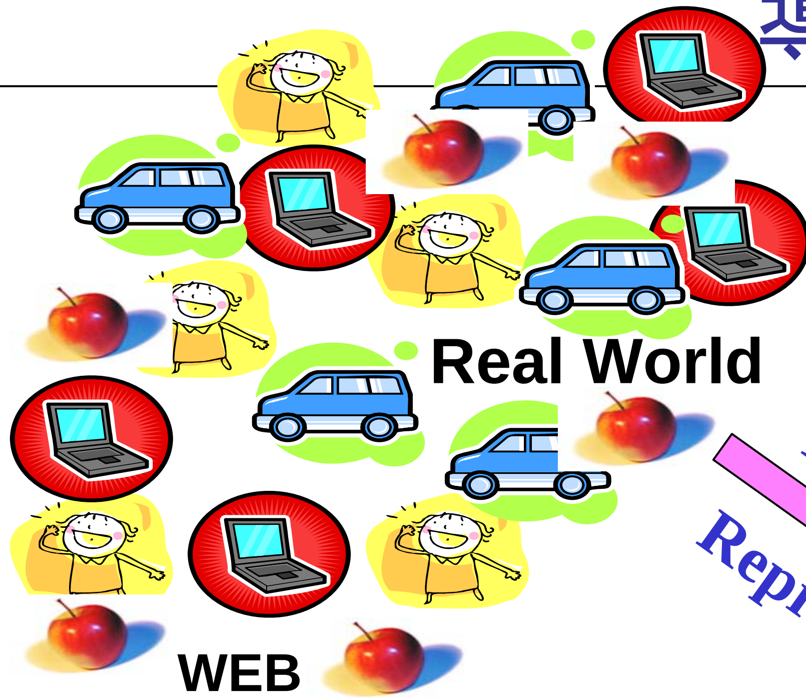
- オブジェクトの移動をサポートするJava機能をベースに:
  - 動的なクラス定義のローディング
  - オブジェクトの逐次化(serialization)
- JavaGoX は現在実行されているオブジェクトは効率よく移民させることができる。
  - 実行スタックをセーブしたり回復したりするコードに直前・直後に挿入する。
  - バイトコード変換により実装した

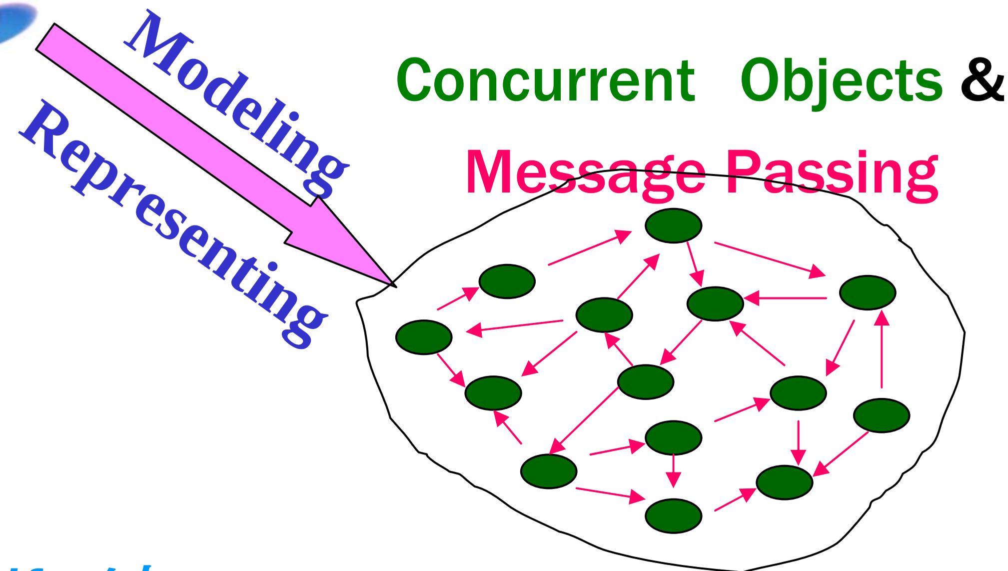
cf. Sakamoto, Sekiguchi, Yonezawa: Bytecode Transformation for Portable Thread Migration in Java, in ASA/MA'00, 2000 for detail.

（最優秀論文賞）

並列オブジェクトを
　　　大量の使用するシステム

並列オブジェクトを
導入した動機に戻って





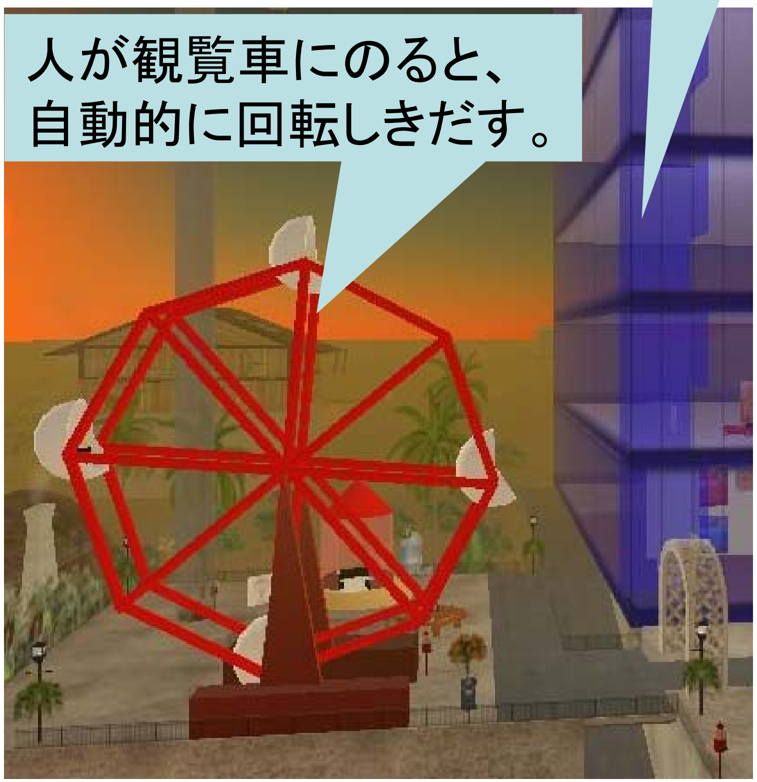
Entities,people,
machines &
their interactios

Linden社の *Second Lifeは* …

並列オブジェクトを考えた「動機」の自然な結果の1つ！

「セカンドライフ」で使われる 並列オブジェクト

Linden社の仮想世界オンラインシステムは
何百万人ものユーザが参加している!



○登場物やアバタは
並列オブジェクトとして
表現・プログラムされてる!!

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

Second Life の中の並列オブジェクト

- Jim Purbrick, Mark LentcznerはOOPSLA07の招待講演：

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

  で、つぎのように述べている:

並列オブジェクトそのもの

  # 登場物とアバタは互いにメッセージをやりとりしながら協調している。

  # 各登場物とアバタは以下のようにプログラムされている:

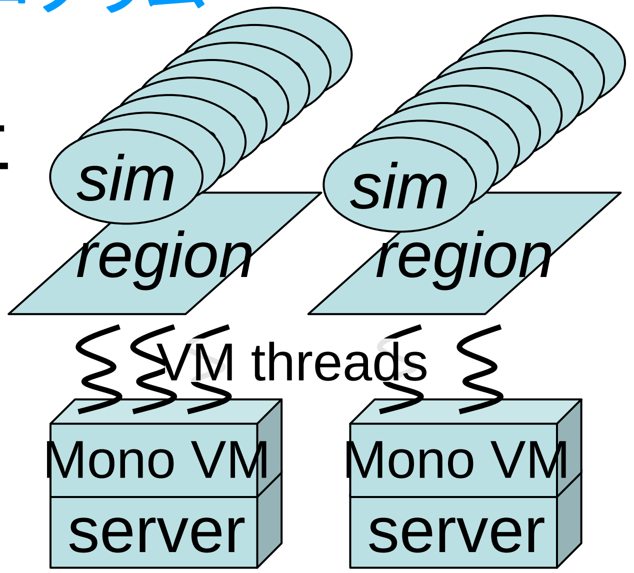
  1. 自分自身の状態をもつ,
  2. 到着するメッセージに反応するメソッドを持っている、
  3. 異なる状態ごとに異なる反応をすることができる、
  4. 自分自身のスレッドをもっている

  – Second Lifeでは、約200万個の登場物がプログラムされ、それらが行動している。

Second Lifeの新しい 実行エンジンは Mono*の上で動く

Second Lifeの新しい実装

- より多数の“sim (simulated objects/Cos)” を動かしたい
  - 1つのregionでつね1000個ものプログラムスクリプトを動かしたい;
- 実行されているSimのregion間に自立的移動 を高速にしたい



***Mono: MS CLIと互換 オープンソースの実行時系**

*Purbrick (babbagelinden)のブログ：Monoでのマイクロスレッド”, May 2006

JavaGoXで開発した、コード変換法がSecond Lifeで使用されている

- 我々のJavaGoX [ASA/MA’00]
  - Javaのバイトコードを変換する手法をベースにしている。これによって、現在JVM上で実行中のオブジェクトを容易にかつ効率よく移動・移民させることができる。
  - Second Life では同様な方法を用いて、新しいMonoベースのスクリプト実行マシン上で、移動・移民を実現している。
  - “regions”間の移動・移民
    - 各regionは1つのサーバーで管理・実行されている。

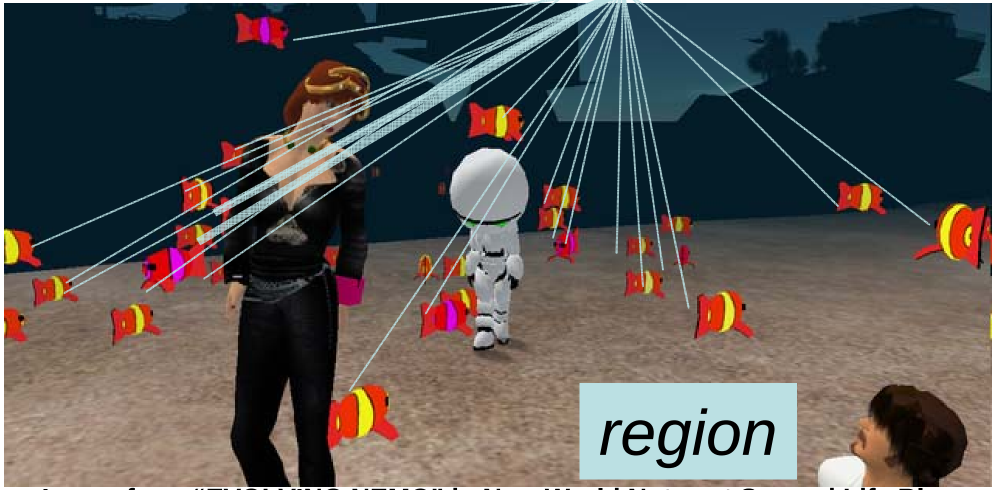


Image from “EVOLVING NEMO” in New World Notes at Second Life Blog (http://secondlife.blogs.com/nwn/2005/06/evolving_nemo.html)

展望

Second Lifeでなぜ並列オブジェクトが使われたか？

- 並列オブジェクトのアイデアが Second Life で使われた理由:
  - 並列オブジェクトが仮想世界の登場物を*直接的にシミュレーションできる、*
  - これによって、

    モデル化が容易になり 、かつ

    並列プログラミングが安全でやさしくなる!

ErlangやRevactorになぜ並列オブジェクトが

- **Erlang:** WEBアプリや
  分散、フォルトトレラント計算用に人気
- Revactor: 並列オブジェクト・アクターモデルのRuby用
  の実装で、WEBアプリで人気
- 両者ともが使っているのが:
  - 非同期方式のメッセージ送信によるコミュニケーションで、
    共有メモリを介する通信を使わない
  - 超軽量スレッドによる、mailboxを介してのsend/receive操作
- なぜか？
  - 「ロック・解除」操作が不要になる！！！
    => **容易で安全は細粒度並列プログラミング!!!**

マルチコアマシンの時代が到来

- 2, 4, 或いは 8 way のマルチコアをノードとするマシンが入手可能になっている
- このようなマシンの性能を極限まで引き出すには、超細粒度の軽量スレッドが手軽に使えて、共有メモリによらない、通信方式が必要になる
- それも僅少なコストで!!

それが今可能になりつつある!!

もっと前にすすめる…

“並列オブジェクト” が享受するものは:

– 自然かつ強力はモデル化能力,

– 容易で安全な 並列性とスレッドの活用・管理

– 超並列軽量スレッドの実装技術、
（例えば、*StackThreads*） が実現されてきた。

– マルチコアアアーキテクチャが実現され皆に使われる.

もっと精密でもっと強力な

物理世界、生態系、生物体、社会構造、組織構造

などの実世界や仮想世界の

モデル化・シミュレーション・プログラミングが可能になる

ご清聴に深謝