言語をつくる

上田 和紀（早稲田大学）

JSSST 2018, August 30, 2018

なぜ言語をつくるのか

- （ソフトウェアでもハードウェアでも）つくることは（論文読みよりも）楽しい
- つくるためのものをつくることはもっと楽しい
- アート，科学，工学 が互いに支えあう接点で仕事ができる
  - デザイン，アルゴリズム，定理，数値（性能等）のすべてを相手にできる
  - Knuth: “The *Art* of Computer Programming” (1968)

どんな言語をつくる（つくった）のか

- 次のような言語を作ってきた
  - 非手続き型（宣言型??）言語
  - 並行プログラミング言語
  - 状態空間探索と検証のための言語
- 既存の言語とは明確に異なる言語
  - “A language that doesn’t affect the way you think about programming is not worth knowing.”
    – Alan Perlis (1922-1990)
- すべて minimalist アプローチ
  - とは言っても処理系は数万～10万行規模

なぜつくるのか

計算とプログラムの本質を理解したい

- ラムダ計算 (1930's) ですら相当に分かっていない
  - "We don't understand substitutions."
    – expert reviewer of RTA2008
  - "Counting and Generating Lambda Terms"
    by Grygiel and Lescanne, 2013
- 既存の枠組みは，根本的な部分から疑い，考え直す価値がある

なぜつくるのか

計算とプログラムの本質を構成的に理解したい

- 設計から実装までゼロから構築すれば，生半可な理解にとどまる可能性が減るであろう
- 自由な発想で展開できる
- プログラミング言語は動いてナンボ
- 日本はものづくりの国
  - “things” + “acts of making” + “Kaizen”

どうやってつくるのか

- 料理に例えると…
  - （料）素材はほとんどあり合わせ
    - 数学や論理学からとってくる
    - 既存のプログラミング言語からとってくる
    - それらの良い性質を活かす
  - （理）新たな使い方，組み合わせ方を開拓
    - “computational interpretation”
  - 「足し算」よりも「引き算」
    - 足し算 ➔ 機能間相互作用を伴うかも

懸念

- 使ってくれる人はいるの？
  - 答１：キラーアプリが出る可能性がある
  - 答２：コンセプトと proof of concept がしっかりしていれば相当に長持ちする
  - 答３：言語の gene pool の維持発展は重要

これまでに作ってきた言語

- Guarded Horn Clauses (1984～)
  - 制約に基づく並行計算
    (constraint-based concurrency; a.k.a. CCP)
  - 関連する言語: CHR, X10, LMNtal
- LMNtal (2002～)
  - グラフ書換え言語とその（並列）モデル検査器
- HydLa (2008～)
  - ハイブリッド制約言語と処理系

森畑先生（プログラム委員長）からの依頼メール

（前略）
上田先生はFLOPS2016でのご講演が大変印象的だったこともあり、私が推薦いたしました。第五世代コンピュータプロジェクトは、1981年生まれの私にとってはほとんど「歴史上の出来事」で、そして「大失敗した」ものだと思い込んでおりました。しかし、上田先生のお話をうかがい、改めて自分の専門的知識に照らし合わせてみれば、少なくとも単純に「失敗」と断言できるものではない、と認識を改めました。

上田先生は第五世代コンピュータプロジェクトの時代から一貫性をもって、プログラミング言語・ソフトウェアの関する独創的な研究されているという印象です。
（後略）

プロジェクト初期から現在までに考えた
「問い」と「教訓」をいくつか紹介します

プロジェクトの評価

- 大規模プロジェクトは多くの評価尺度を伴う．
  - 学術的（ここはさらに多数），
    経済的，社会的，人的，…
  - 国際的，国内的，…

$(v_1, \ldots, v_n) \geq 0$??? (type error!)

- 実プロジェクトは代案との比較がほぼ不可能
- 学術的評価，特にコンセプトの評価は25年後でも時期尚早

第五世代プロジェクトの文献

- Logic/Constraint Programming and Concurrency: The Hard-Won Lessons of the Fifth Generation Computer Project.
  Science of Computer Programming, **164** (2018), pp.3-17
  - https://authors.elsevier.com/a/1XYZoc7X4nQNp
    （９月末ごろまで無料ダウンロード可能）

- The Fifth Generation Project: Personal Perspectives.
  CACM, **36**(3) (1993) (D.H.D. Warren & E. Shapiro, eds.)
  - 渕一博，Robert Kowalski，古川康一，上田和紀，
    Ken Kahn，近山隆，Evan Tick が個人の立場から寄稿

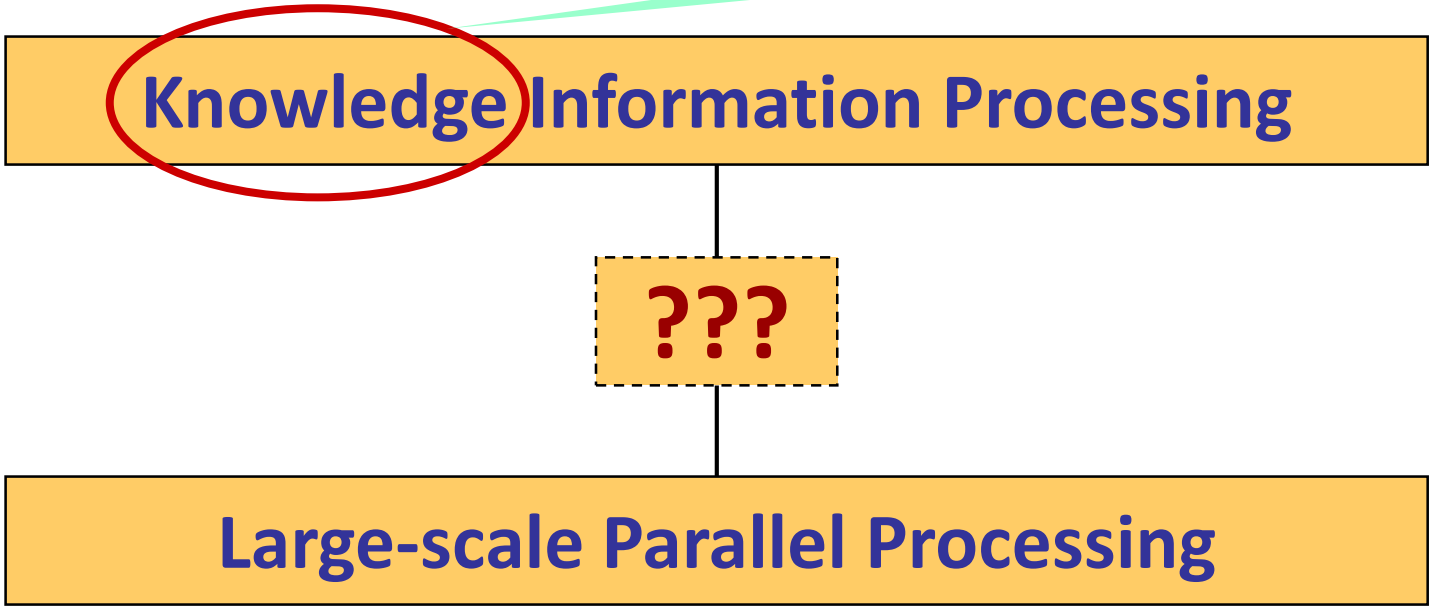
第五世代プロジェクトの文献

- AITEC・ICOTアーカイブズ：わが国の先端情報技術開発
  (2005) http://www.ueda.info.waseda.ac.jp/AITEC_ICOT_ARCHIVES/
- 『第五世代コンピュータの並列処理』
  瀧和男編，bit別冊，共立出版，1993.
  （編著者と共立出版の了解を得て以下に掲載）
  http://www.ueda.info.waseda.ac.jp/~ueda/documents/
- 渕一博記念コロキウム『論理と推論技術：四半世紀の展開』
  (2007) http://www.ueda.info.waseda.ac.jp/fuchi-colloquium/
- 『渕一博：その人とコンピュータサイエンス』
  近代科学社，2010.
  - 林晋：情報技術の思想家 渕一博（第１章）
- 特集：『第五世代コンピュータと人工知能の未来』
  人工知能，Vol.29, No.2, 2014.

第五世代プロジェクト (1982–1993) (cf. JSSST 1983-)

- ICOT研究員40～100名，540億円
- **チャレンジ目標:** 知識情報処理と並列処理の架け橋となる方法論とシステムの開拓

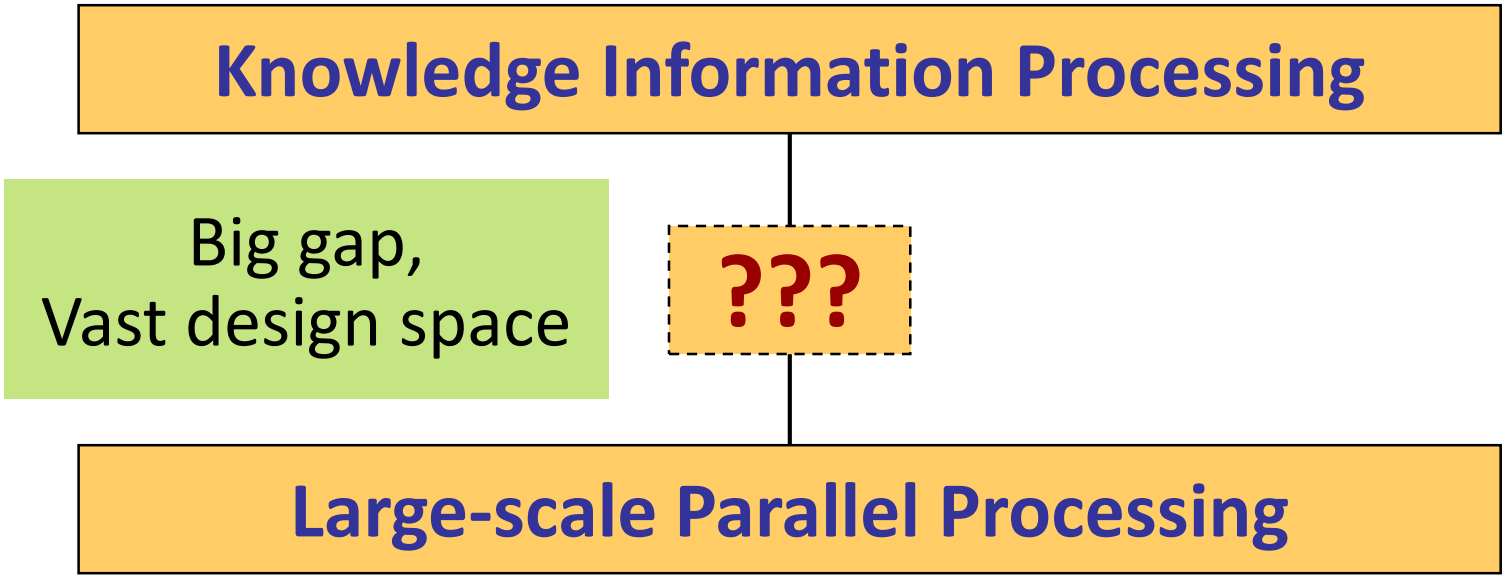
物理記号系仮説の時代



cf. Computer architecture as the hardware/software interface

第五世代プロジェクト (1982–1993)

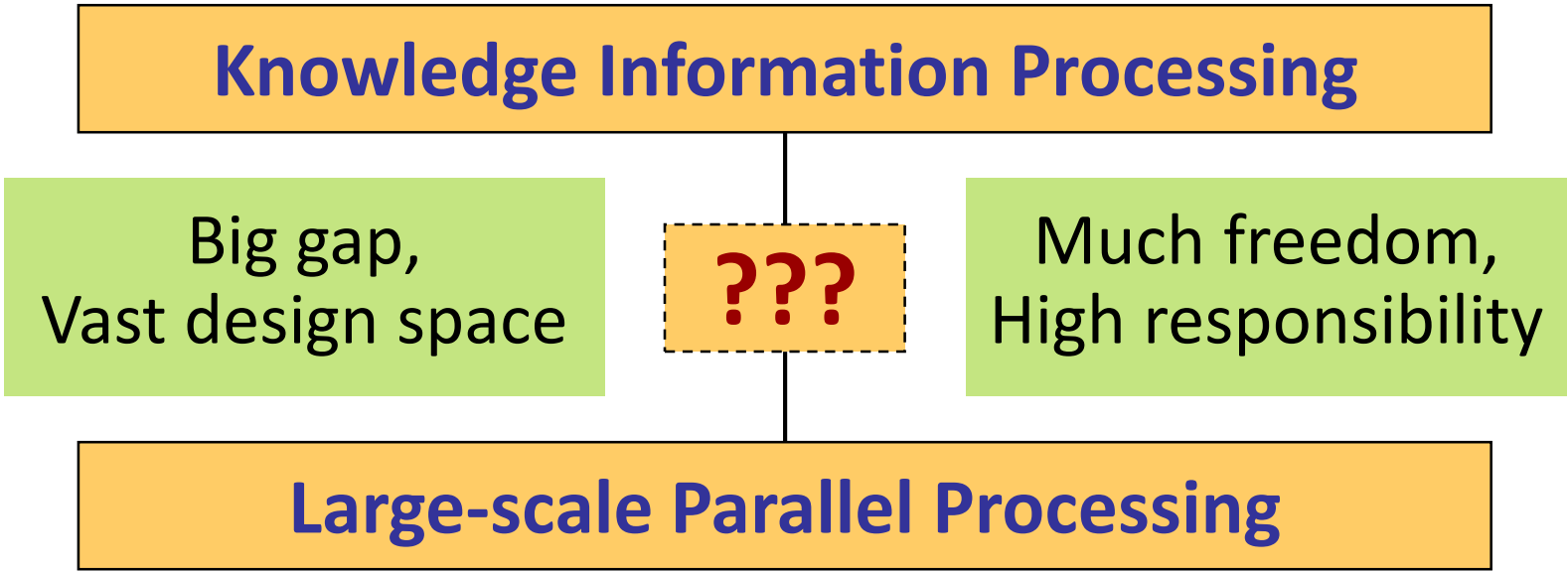
- ICOT研究員40～100名，540億円
- **チャレンジ目標:** 知識情報処理と並列処理の架け橋となる方法論とシステムの開拓



cf. Computer architecture as the hardware/software interface

第五世代プロジェクト (1982–1993)

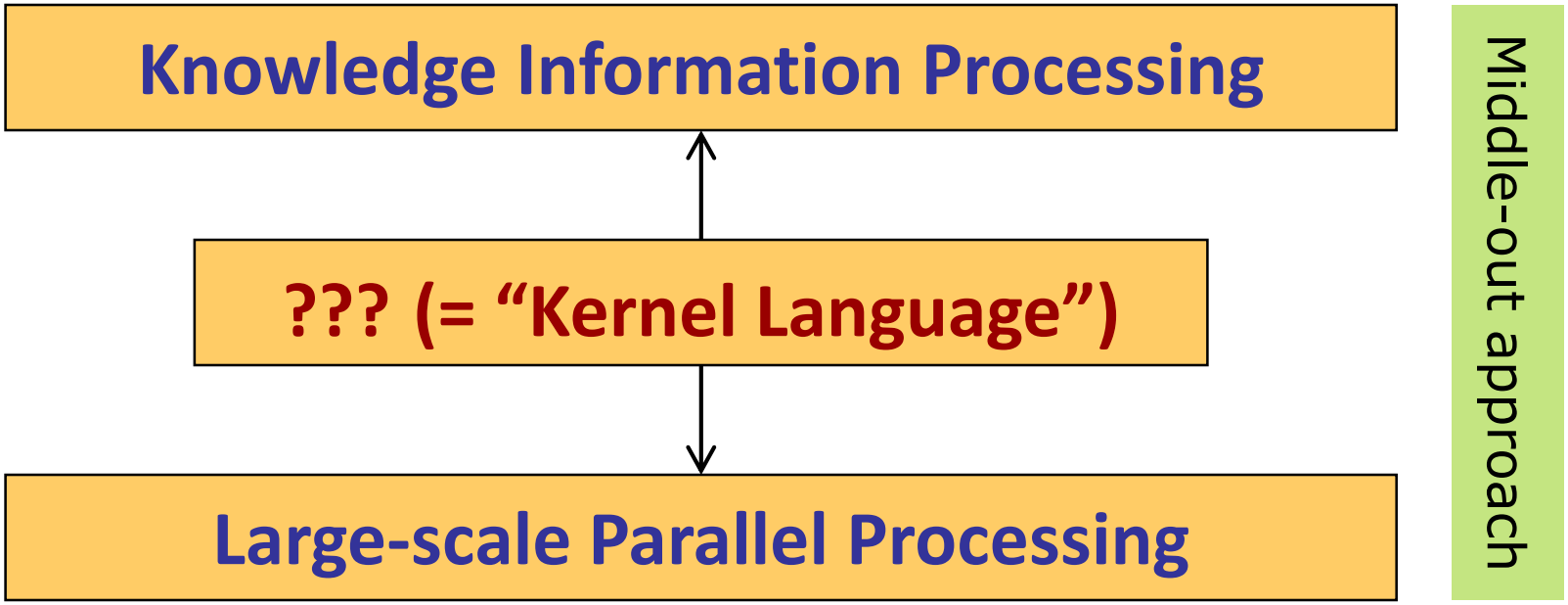
- ICOT研究員40～100名，540億円
- **チャレンジ目標:** 知識情報処理と並列処理の架け橋となる方法論とシステムの開拓



cf. Computer architecture as the hardware/software interface

第五世代プロジェクト (1982–1993)

- **チャレンジ目標:** 知識情報処理と並列処理の架け橋となる方法論とシステムの開拓
  - ➔ **（第五世代）核言語**の設計と実装が中心テーマに



だれがどこで推進したか

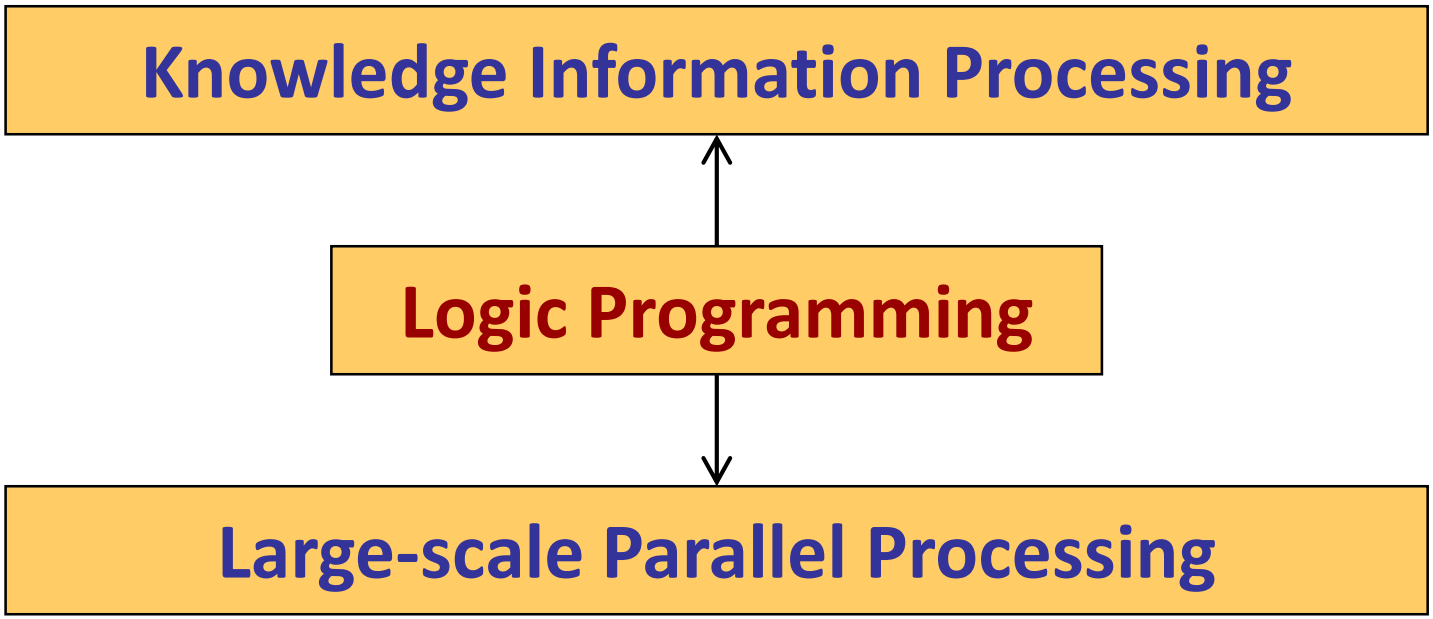
ICOT研究所 21F (+ 計算機室等)

ICOT同窓会 (2014年1月)

作業仮説 (Working Hypothesis)

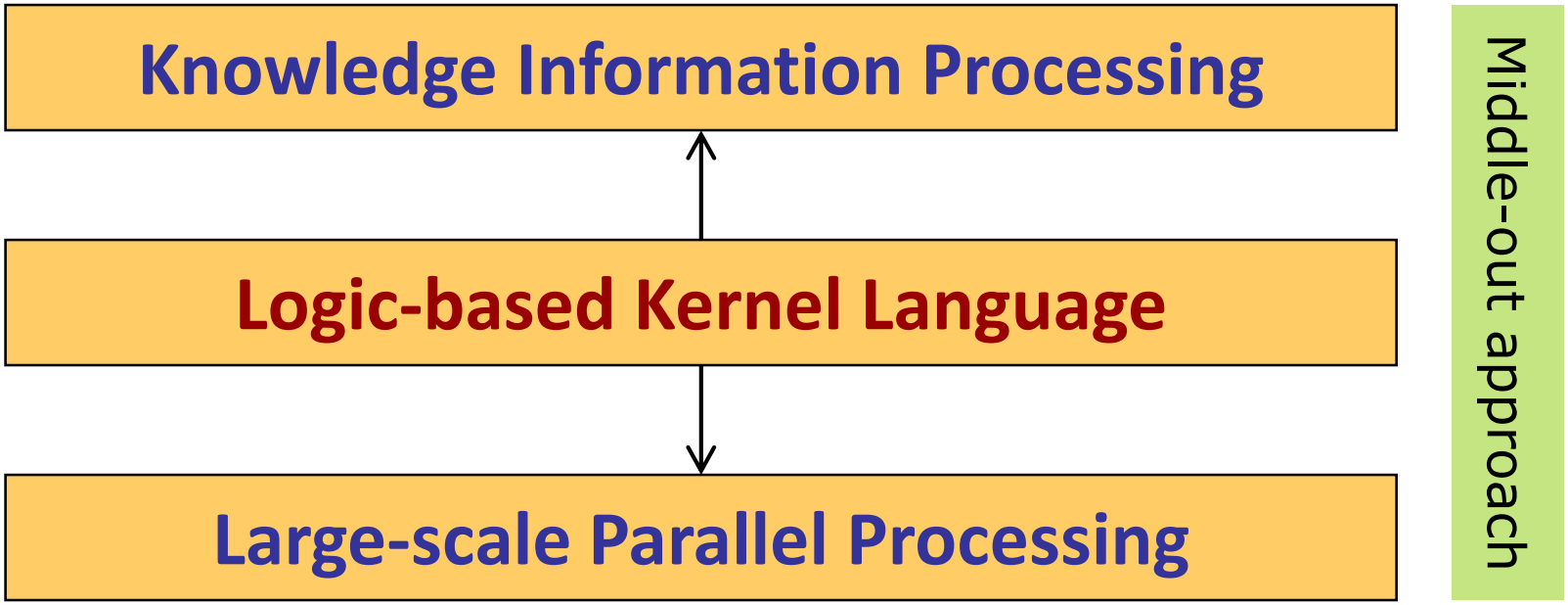
- 広い意味での「論理プログラミング」が選ばれた
  - 理由は二つ

Research question here



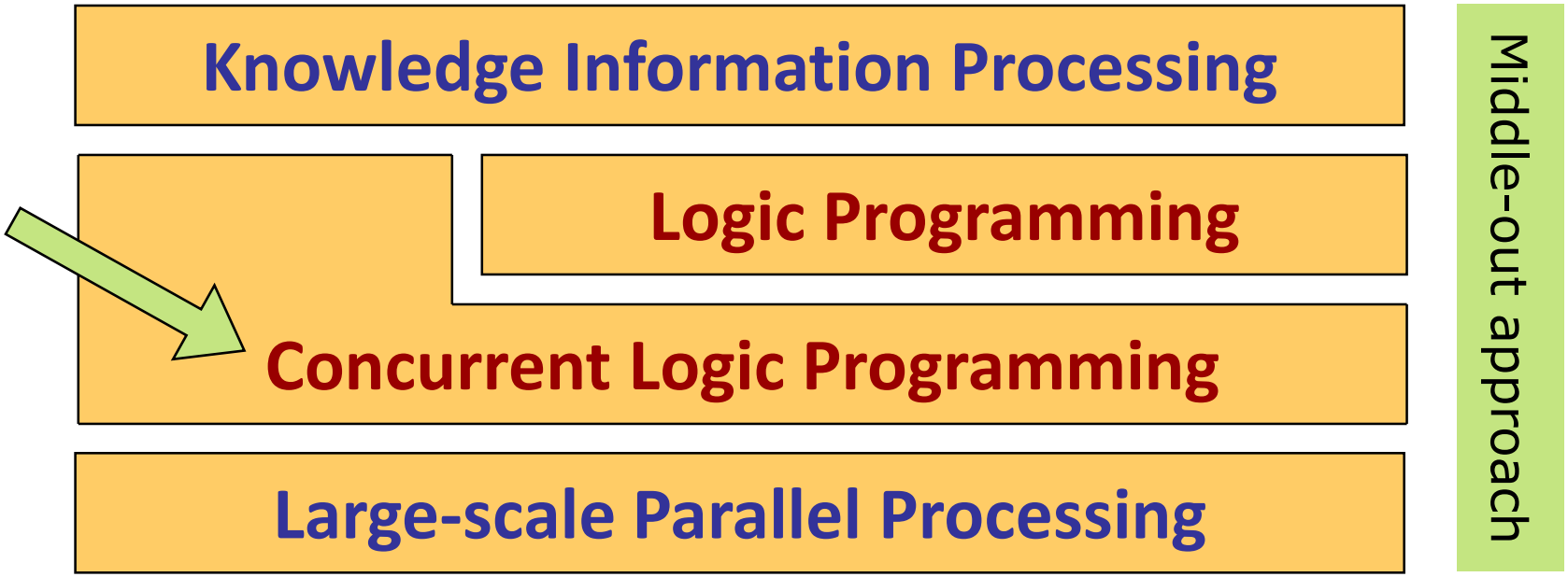
第五世代核言語 (kernel language)

- **言語づくりはプロジェクトの核であった**
  - **KL0**: Sequential kernel language for startup
  - **KL1**: Parallel kernel language for systems and apps
  - (KL2: Knowledge representation language)



プロジェクト終了時の形 (1993)

- いくつかの成果
  - Kernel Language (GHC/KL1)
  - Parallel OS (PIMOS) totally written in KL1
  - Parallel Inference Machine (PIM with 512 & 256PEs)
  - Model Generation Theorem Prover (MGTP)



ハードウェアの開発と利用

渕 一博 (1936-2006)

D.H.D. Warren

AITEC-ICOT
Archives
DVD, 2005

PSI-I (Dec. 25, 1983)
35KLIPS for KL0
80MB, 100 copies
SIMPOS in ESP
+ many apps

PSI-II (Dec. 1986)
330 (400) KLIPS for KL0
with 6.67MHz, WAM, 300 copies

ハードウェアの開発と利用

渕 一博 (1936-2006)

D.H.D. Warren

AITEC-ICOT
Archives
DVD, 2005

PSI-I (Dec. 25, 1983)
35KLIPS for KL0
80MB, 100 copies
SIMPOS in ESP
+ many apps

PSI-II (Dec. 1986)
330 (400) KLIPS for KL0
with 6.67MHz, WAM, 300 copies

ハードウェアの開発と利用

内田俊一

Multi-PSI (Mar. 1988)
64 PSI-II's in 2D mesh
5MLIPS for KL1, 6copies
PIMOS (standalone, multi-user OS in KL1, 44KLOC in 0.5years)

PIM/m (1992)
256 PEs in 2D mesh (256 PEs)
200 MLIPS, PIMOS (200KLOC) in KL1

AITEC-ICOT Archives DVD, 2005

核言語の形成過程

[CACM March 1993 issue]
[Sci. Comput. Program. 2018]

KL1設計タスクグループ (1983-04〜)

- 古川康一（リーダー）
- 設計要件 (cf. Prolog)
  - 汎用言語
  - 並列アルゴリズムの記述
  - OSの記述
  - メタプログラミング

Keith Clark　Ehud Shapiro

- 並行論理プログラミングの研究者を招聘して日夜議論 (1983-10)
  - Keith Clark, Steve Gregory (PARLOG)
  - Ehud Shapiro (Concurrent Prolog)

なぜ論理型言語だったか（所内討議スライド, 1984）

5Gプロジェクトの目標

「1990年代における真に汎用なコンピュータの開発」

- 数値計算ではなく 記号計算
- データ処理ではなく 知識処理
- 超並列アーキテクチャによる高速化
  （非ノイマン・コンピュータ）
- 新しい ソフトウェア科学・工学の追求

Prolog

= Lisp − α + β

+ small Relational DB

α : Functional Argument

β : Imcomplete Data Structure

$\beta >> \alpha$

並行論理プログラミング (1980～)

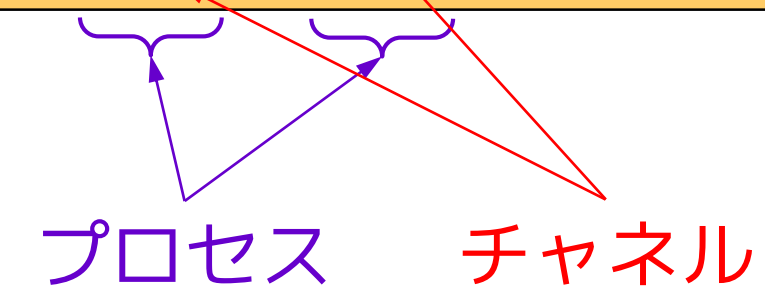
- 構文: ガード付き節 (cf. Guarded Commands)
- 意味 (cf. Prolog) :
  - データフローに基づく同期（＝計算の半順序）
    ＋ 非決定的選択 (don’t-care nondeterminism)

```
p(ok) :- true | ... .
q(Z)  :- true | Z=ok.

:- p(X), q(X).
```

（受信, ask）

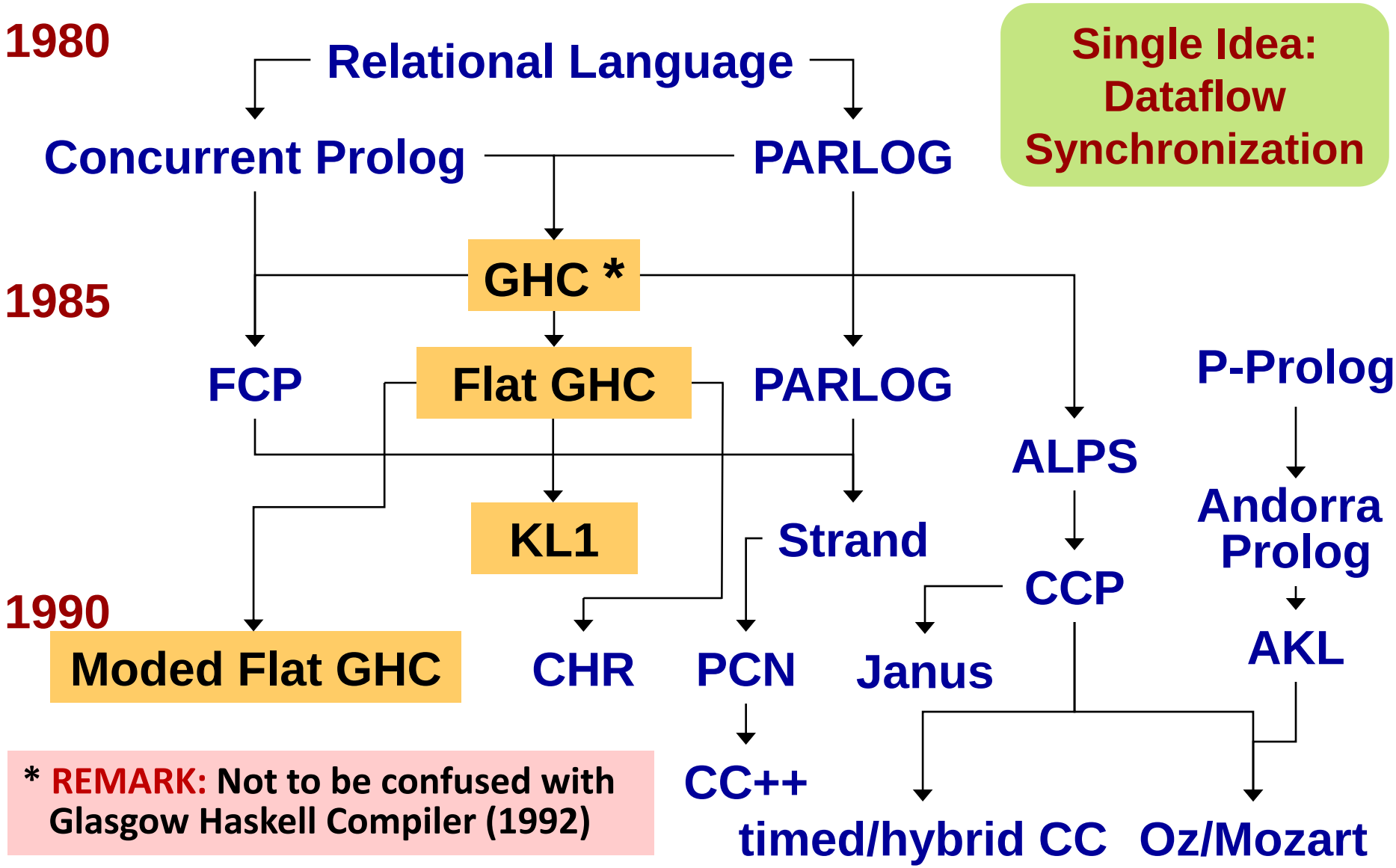
（送信, tell）



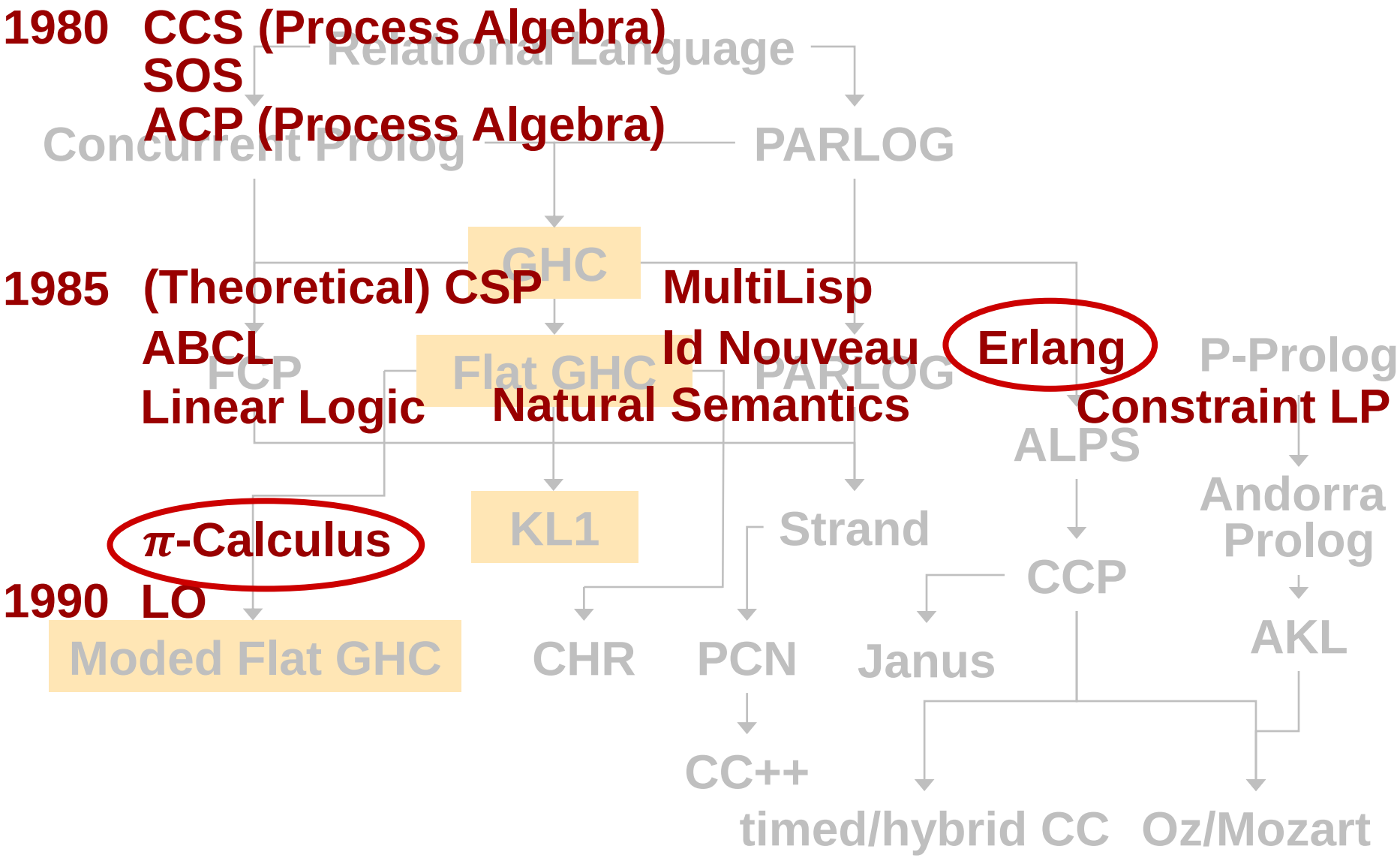
プロセス

生成，消滅，変化，分裂，合体，送受信などをルールにしたがって自律的に行う実体

並行論理・制約プログラミングの初期の歴史



Early History of Constraint-Based Concurrency



当時の議論・雑談から (1983)

- 鈴木則久（当時東京大学）
  “Building the entire system without resort to side effects is the FGCS project’s right way to go.”
- 内田俊一 (ICOT):
  “The computational model of KL1 should not assume any particular granularity of underlying parallel hardware.”
  (= Kernel Language should embrace as fine-grained concurrency as possible.)

当時の議論・雑談から (1983)

- Ehud Shapiro (Weizmann Institute)
  （KL1要求仕様案を見て）*“Too many good features.”*

➔ タスクグループの research question が定まる

“What’s the minimum set of language constructs that turn Logic Programming into an expressive concurrent language?” (Occam’s Razor, cf. CSP/Occam)

当時の議論・雑談から (1983)

- Ehud Shapiro (Weizmann Institute)
  （KL1要求仕様案を見て）*“Too many good features.”*

  ➔ タスクグループの research question が定まる

> *“What’s the minimum set of language constructs that turn Logic Programming into an expressive concurrent language?”* (Occam’s Razor, cf. CSP/Occam)

- 1984年2月には，Concurrent Prolog が KL1 設計のための（詳細化された）作業仮説となる
  - π 計算の意味での動的プロセス（チャネル）構造が５年以上前に実現されていた

Research question への解答

- 先輩言語の解答：
  - 入出力モード概念の導入 (PARLOG)
  - 変数の capability 概念の導入 (Concurrent Prolog)
- **Guarded Horn Clauses (GHC)** (Dec. 1984) [LNCS 221]:
  - 単一化（によるデータフロー）の制限
  - 新たな構文要素なしに動的チャネルや返信箱つきメッセージを実現
  - ハードウェアグループも歓迎，KL1の新たな作業仮説に採用（1985年初頭）
  - Lesson: *simplicity was the key to the consensus.*

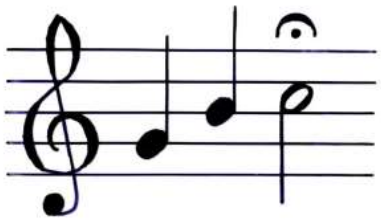
Guarded Horn Clauses [LNCS 221]

- プロトタイプ処理系 (DEC-10 Prolog) は１日半で作成
  - Prolog 上のConcurrent Prolog コンパイラを翻案 [SLP 1985]（まだ動きます．公開中）

    www.ueda.info.waseda.ac.jp/~ueda/software/ghcsystem-swi.tgz
  - Gerard Huet (INRIA) は CAML で数日で記述 (1988)
- すぐに Flat GHC サブセットに移行 (1985中ごろ)
- 論理型言語の探索機能 (proof search) の実現法を提案 [ICLP 1986]（第２回大会高橋奨励賞）

当時のOHPスライド (1986頃)

GUARDED HORN CLAUSES



KAZUNORI UEDA

(ICOT)

Semantics of GHC

H_1 :- $\cdots$ G_{1j} $\cdots$ | $\cdots$ B_{1j} $\cdots$.
H_2 :- $\cdots$ G_{2j} $\cdots$ | $\cdots$ B_{2j} $\cdots$.
$\vdots$ $\vdots$ $\vdots$

Can be tried in //, but cannot instantiate* the caller.

Can instantiate the caller, but only one selected clause is allowed to do so.

Selects one of the clauses whose guard has succeeded (commitment)

(The other clauses cannot contribute to successful computation.)

* Instantiate: making more specific

p(X, Y) → p(1, Y) instantiated
p(X, Y) → p(A, B) not instantiated
p(X, Y) → p(A, A) instantiated

当時のOHPスライド (1986頃)

DESIGN PRINCIPLES OF GHC

1. Parallelism

- It must be a parallel language 'by nature', not a sequential language augmented with constructs for parallelism,
  - — to have a clearer semantics, and
  - — to disallow inessential sequentiality to creep in.
- Introduction of sequentiality is considered as an optimization to meet the current computer architectures.
- We have to allow even possibly useless computation.

2. Generality

- It must be a general-purpose language which can express important concepts in parallel programming.
- It must be general also in that no specific implementation scheme is assumed a priori.

3. Simplicity

- It must be a simple language because of the shortage of our experience both in the theoretical and the practical aspects of parallel programming (languages).

4. Efficiency

- It must be an efficient language which allows fast execution of simple programs at least under the current computer architectures.
  - — cf. generality (2.)
- Sequential implementation is more than a prototype.
- Efficiency may interfere with generality and simplicity, but a general language could be subsetted for more efficient execution of a specific class of programs.

GHC から KL1 へ

- GHC は並行言語モデル
- KL1 は（OSも書ける）並列言語
  - プロセスからプロセッサへのマッピング
  - プロセスの保護

- Lesson: *separation of concerns*
  - 並行処理　vs.　並列処理
  - 並行処理　vs.　探索　(cf. multiparadigm langs)
  - 簡約　vs.　通信
    （「通信の不可分操作」論争, cf. π-calculus)

制約に基づく並行計算

— what we developed in mid 1980s, in retrospect

制約に基づく並行計算

- **Concurrent Constraint Programming** (CCP, late 1980's)
  - 制約論理プログラミングに触発された並行論理プログラミングの定式化
  - プロセス間通信の論理的解釈 (**Ask** / **Tell**)
  - データ領域の抽象化・一般化
- 二つの構成要素
  - 単一代入 **(write-once) チャネル**
  - **（データ）コンストラクタ**

(cf. CCS, CSP, π, etc.)

単一代入チャネル

- 論理（型言語の）変数の応用
- 書き込みは１回だけ
  - チャネルの値についての部分情報（制約）を (unificationで) ***tell*** (publish) する
    - E.g., `tell S=[read(X)|S' ]`
- 読み出しは非破壊的
  - ある部分情報が帰結 (entail) できかを ***ask*** する (*matching*)
    - e.g., `ask ∃A ∃S'(S=[A|S' ])`
  - π 計算の *input* と *match* にほぼ相当

制約概念に基づく通信機構

- 単一代入チャネルによる通信は以下の機能を実現していた (cf. π-calculus)
  - チャネルを使ってチャネルを送る
  - 返信箱つきメッセージを送る
  - チャネルを相互結合する (fusion)
- データ構造の non-strictness を最大限利用
  - 制約ベース = *computing with partial information*
- 共有変数は使うが，本質は，当事者間のみが知る局所変数を使った peer-to-peer の通信

デモ

KLIC (KL1-to-C translator)

- 第五世代後継プロジェクトで開発 (1993-1994, 28億円)
- 今世紀初頭まで上田研で保守，その後は放置
- 汎用メニーコアマシンの急速な普及
  - 10,000円/コア (e.g., Ryzen PC)
  - KLIC復活作戦実施 (2016)
  - 今のシングルコア実行は，20年前の10コア実行よりも10倍高速
- Lesson: *Old software is lightweight and fast. Why not keep it alive?*

その後の技術

並行論理プログラミングから生まれたもの

- **Concurrent Constraint Programming** (late 1980's)
  - 制約論理プログラミングに inspire された一般化
  - 通信機構の論理的解釈 (**Ask** / **Tell**)
  - データ領域の（有限木以外への）一般化
- **CHR (Constraint Handling Rules)** (early 1990's)
  - ゴールの多重集合の書換え系へ拡張
  - 多くの新たな応用（制約ソルバ等）
- **Timed / Hybrid CCP** (early-mid 1990's)
  - 時間，デフォルト，連続変化概念の導入
  - 時間／ハイブリッドシステムの高水準言語へ

並行論理プログラミングから生まれたもの

- **高性能並列計算と Grid のための言語** (early 1990's -)
  - PCN, CC++, HPC++, swift-lang

from Ian Foster @ ANL

> Dear Ueda-san:
> The wonders of Google Scholar citation alerts led me to your recent paper on FGCS, which I enjoyed reading.
> …
> While PCN and CC++ are long gone, we continue to work with Swift (swift-lang.org), which is really CLP in another guise.
> My best wishes from Chicago.

- **X10** (mid 2000's)
  - IBM's solution to HPC languages

グラフ書換え言語への展開 (2002)

- KL1の経験：同じことを書くのに二通りの方法がある
  - プロセス（構造）とデータ（構造）
    cf. 関数名とコンストラクタ，述語と関数記号
  - Research question: 一本化できないか？

➔ データ構造をなくしても並行プログラムは書ける

  - プロセスの多重集合の書換え言語へ
  - 単一代入変数（論理変数）は無代入(?)変数へ退化

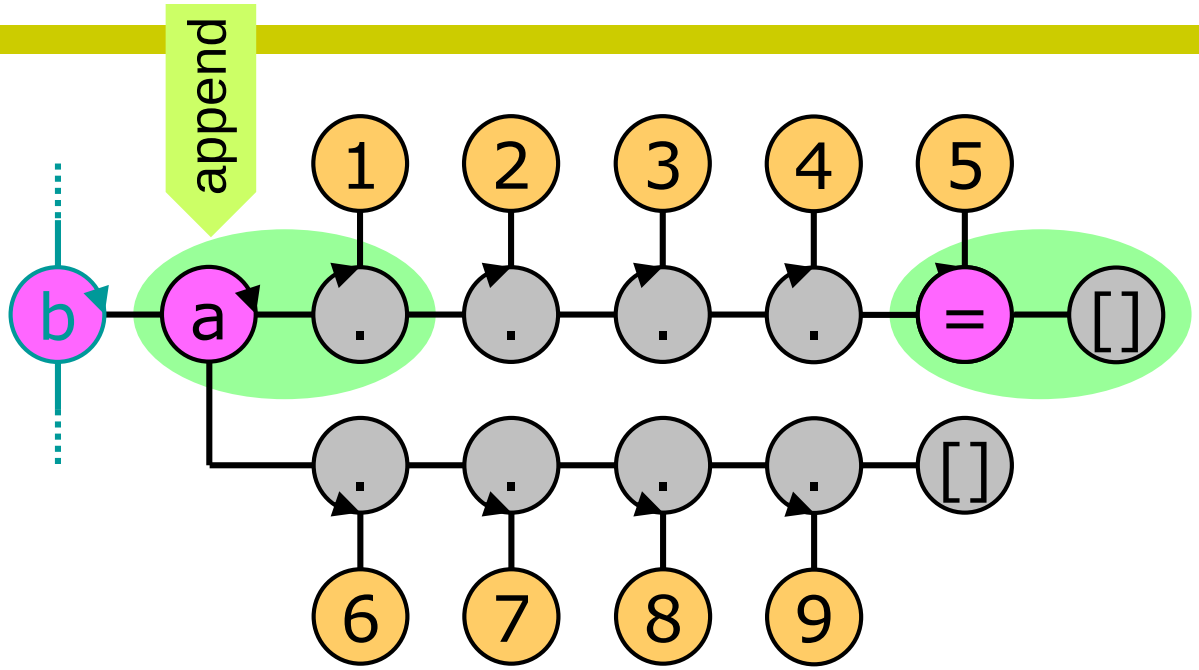
➔ グラフ書換え言語として解釈

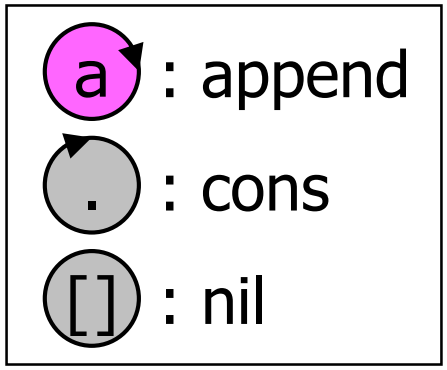
  - **L**inks, (first-class) **M**ultisets, **N**odes

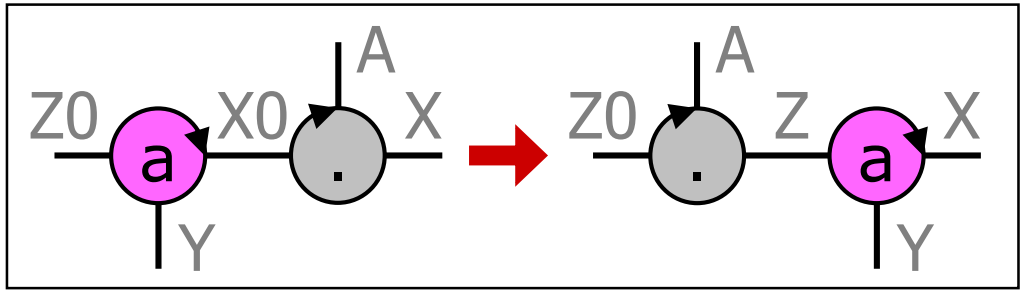
http://www.ueda.info.waseda.ac.jp/lmntal/

https://github.com/lmntal

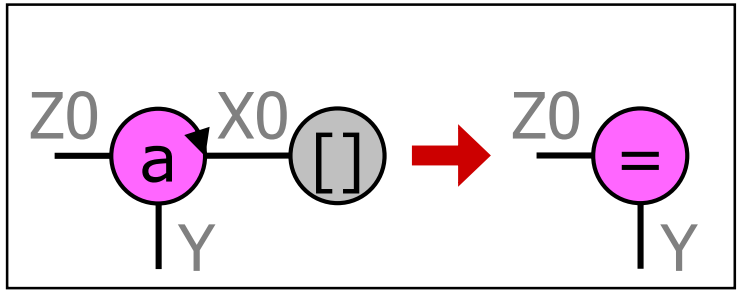
LMNtal でのリスト連結（第19回大会高橋奨励賞）





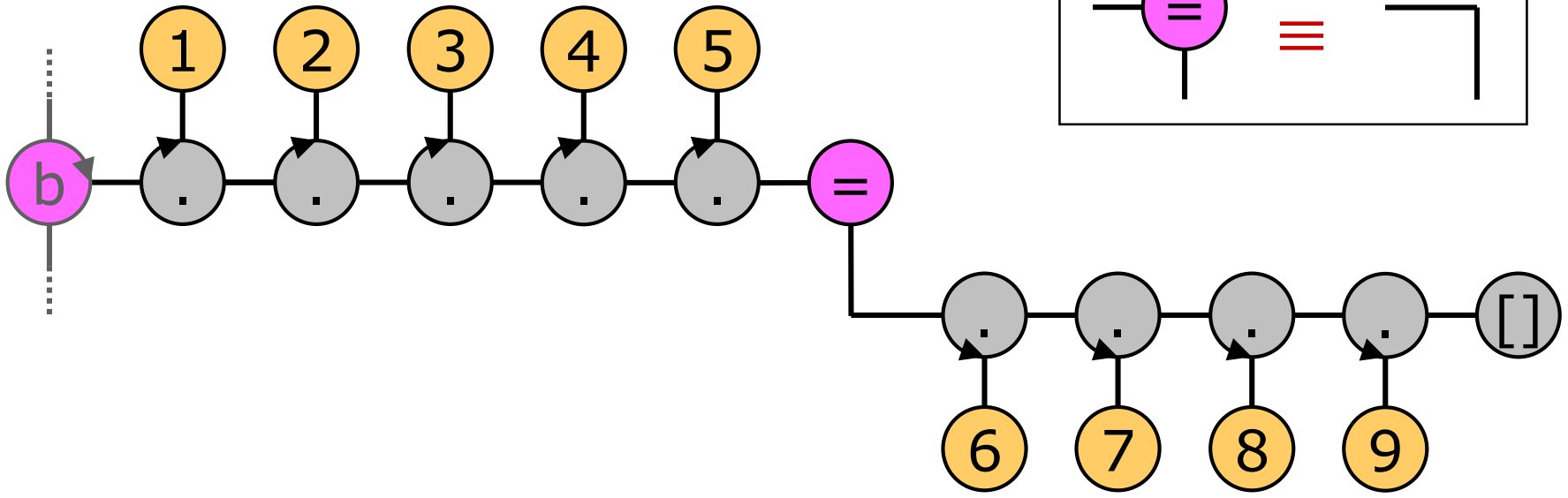


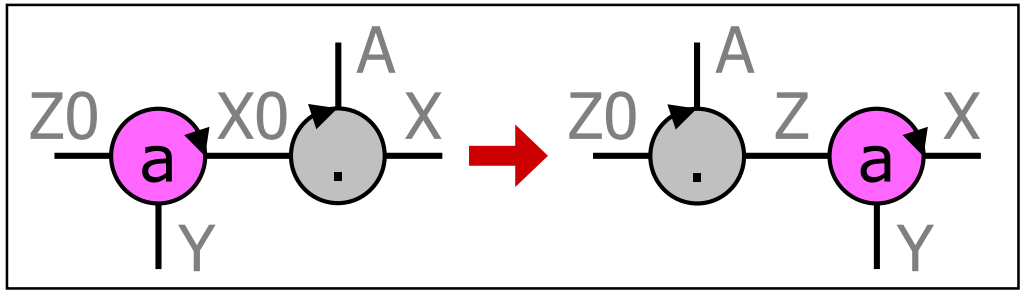
a(X0,Y,Z0), '.'(A,X,X0) :- '.'(A,Z,Z0), a(X,Y,Z)

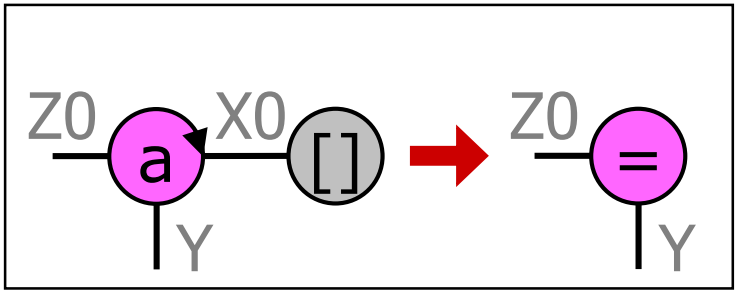


a(X0,Y,Z0), '.'(X0) :- Y=Z0

LMNtal でのリスト連結（第19回大会高橋奨励賞）





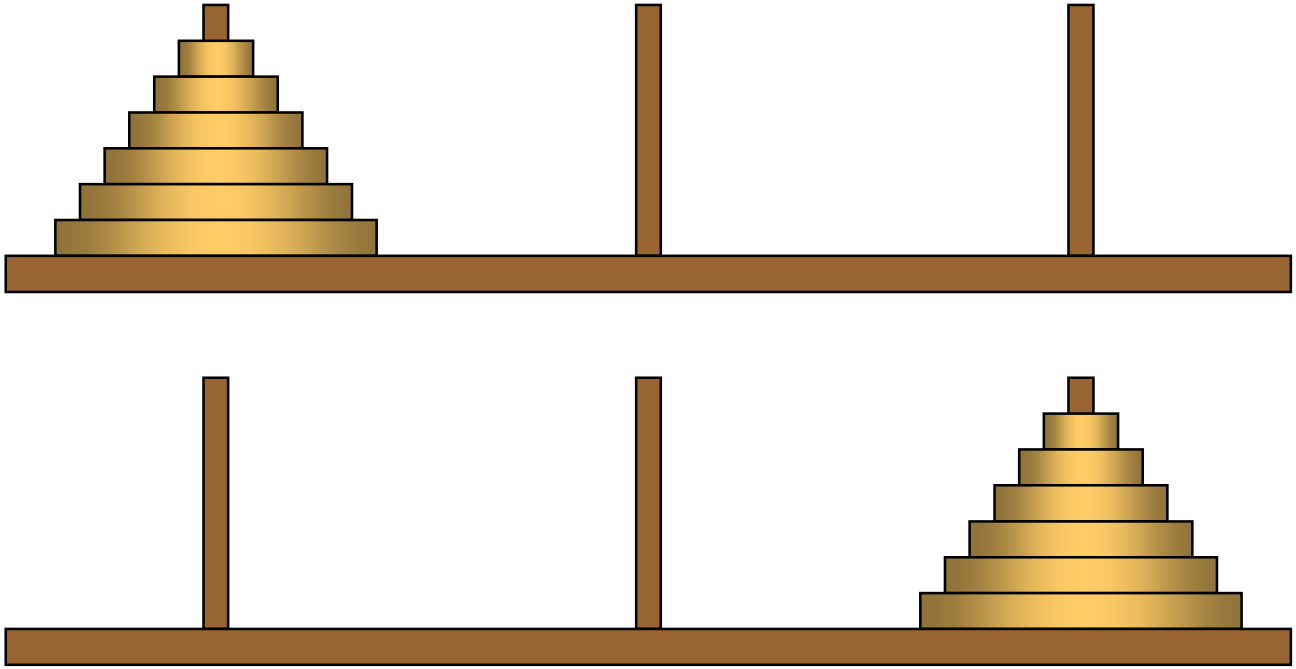


a(X0,Y,Z0), '.'(A,X,X0) :- '.'(A,Z,Z0), a(X,Y,Z)

a(X0,Y,Z0), '.'(X0) :- Y=Z0

LMNtal でのハノイの塔

```
poles(p([1,2,3,4,5,6,99]),p([99]),p([99])).
```



```
P1=p([$h1|$t1]), P2=p([$h2|$t2]) :- $h1<$t2 |
   P1=p(T1), P2=p([$h1,$h2|$t2]).
```

グラフ書換え言語への展開

- 多様な計算体系のエンコードを実現**（当初の想定外）**
  - λ 計算だけで 3 種類作成（細粒度版x2・粗粒度版）
- 並列LTLモデル検査器の形で状態空間探索を実現**（当初の想定外）**
  - グラフ同型性判定を伴う
  - 20億状態までスケール
- 多対多の書換え = 書換え規則の両辺を入れ替えても構文的に正しいプログラム（対称性）
  - 応用例：構文生成器と構文解析器

ハイブリッド制約言語への展開 (2008)

- ハイブリッドシステムは離散システムや連続システムよりも複雑で難しい（∵ 両方の要素＋α）
  - 例：ハイブリッドオートマトン
  - 微分方程式 ＋ 離散変化条件 ➔ 状態概念が必要かは明らかでないが，制約概念は必要

Research Question 1: 複雑さを最小限にしたい．（情報系以外の人にも通じる）数学と論理の記法に最低限何を加えたらモデリング言語になるか？

Research Question 2: 制約処理の基本演算は無矛盾性判定．これだけでモデリング言語が構築できるか？

- 例：条件判定は無矛盾性判定に帰着可能

ハイブリッドシステムのモデリングと計算

- Key issue
  = modeling of, and interfacing with, the *physical* world

Physical systems

$$\frac{d^2x}{dt^2} = 10$$

- Continuous (+ discrete) domain
- Math with differential (+ algebraic) equations
- Time

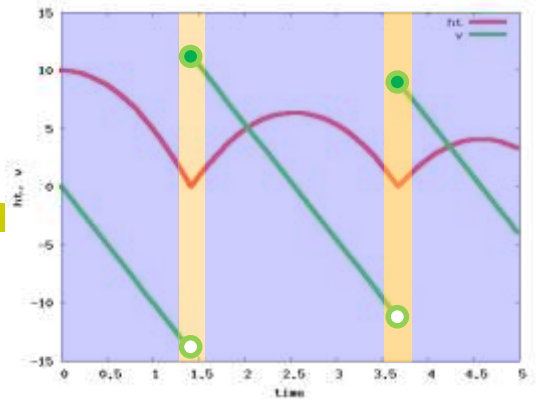
Computer systems

$$x_{t+1} = 1 - x_t$$
$$y_{t+1} = 2\,y_t$$

- Discrete domain
- Programming languages
- Algorithms
- Abstraction

How to reconcile them with computing abstraction of physical systems?

HydLaによるbouncing ball



$$\mathrm{INIT} \Leftrightarrow 9 < ht < 11 \wedge ht' = 0.$$
$$\mathrm{PARAMS} \Leftrightarrow \Box(g = 9.8 \wedge c = 0.5).$$
$$\mathrm{FALL} \Leftrightarrow \Box(ht'' = -g).$$
$$\mathrm{BOUNCE} \Leftrightarrow \Box(ht- = 0 \Rightarrow ht' = -c \times (ht'-)).$$

rules

$$\mathrm{INIT}, \mathrm{PARAMS}, (\mathrm{FALL} \ll \mathrm{BOUNCE}).$$

guard

priority

- 各時点で，制約の優先度を守りつつ，極大無矛盾な制約集合を採用
- 状態は computational interpretation の中で出現

その他の文献

- Concurrent Logic/Constraint Programming: The Next 10 Years
  In *The Logic Programming Paradigm: A 25-Year Perspective*, Springer, 1999, pp. 53-71.
- Logic Programming and Concurrency: a Personal Perspective
  The ALP NewsLetter, 19(2), 2006 (6 pages).
- High-level Programming Languages and Systems for Cyber-Physical Systems
  Halmstad Summer School on Cyber-Physical Systems (HSSCPS, Youtube)

まとめにかえて ー 最近考えていること

- 帰納的な議論がしにくい対象は研究テーマの宝庫
  - プロセス網，グラフ，λ，…
- 一つのプログラミング言語が実は複数の言語からなることが多い
  - 実行の記述＋型の記述＋検証仕様の記述
  - ばらばらでよいのか？
- 部分情報と記号実行も研究テーマの宝庫
  - 実行とプログラム解析の統合に向けて
  - 外延と内包の関係の見直し

Should computing paradigms change?

20th century	21st century
von Neumann architecture + sequential computation	multi-core / clusters / Grid / distributed / embedded / molecular / ...
◆ Turing Machines (computability) ◆ RAM model (complexity) ◆ λ-calculus (programming languages) ◆ Floating point arithmetic (numerical analysis)	

Thanks for the attention!