21世紀のチューリングマシン
— 多様化した計算概念の統合

logic, concurrency, constraints, X-calculi

Kazunori Ueda, Waseda University

(Joint work with many colleagues)

October 20, 2007

Computing paradigms must 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)	?

Computing paradigms must ch

Concurrency Everywhere!

20th century	21 entury
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)	What to teach at Universities?

The FGCS Project (1982 - 1993)

- **The Challenge:** Bridging Knowledge Information Processing and Parallel Processing

Knowledge Information Processing

??? (= "Kernel Language")

Parallel Computer Architecure

The FGCS Project (1982 - 1993)

- **Working Hypothesis:** *Logic Programming*

Knowledge Information Processing

Logic Programming

Parallel Computer Architecure

The FGCS Project (1982 - 1993)

- **Outcome as of 1993:**

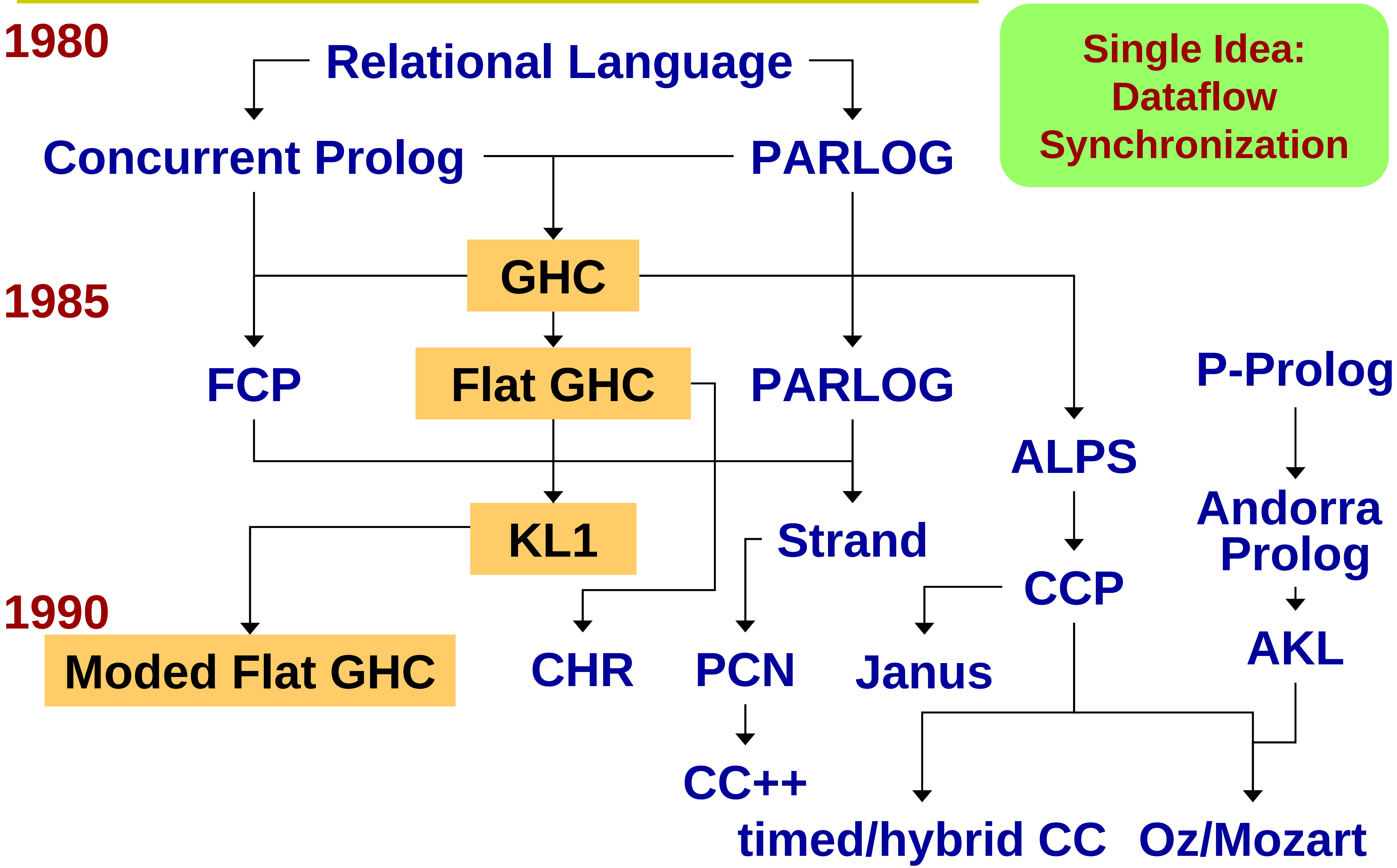
Knowledge Information Processing

Logic Programming

Concurrent Logic Programming

Parallel Computer Architecure

Early History of Constraint-Based Concurrency



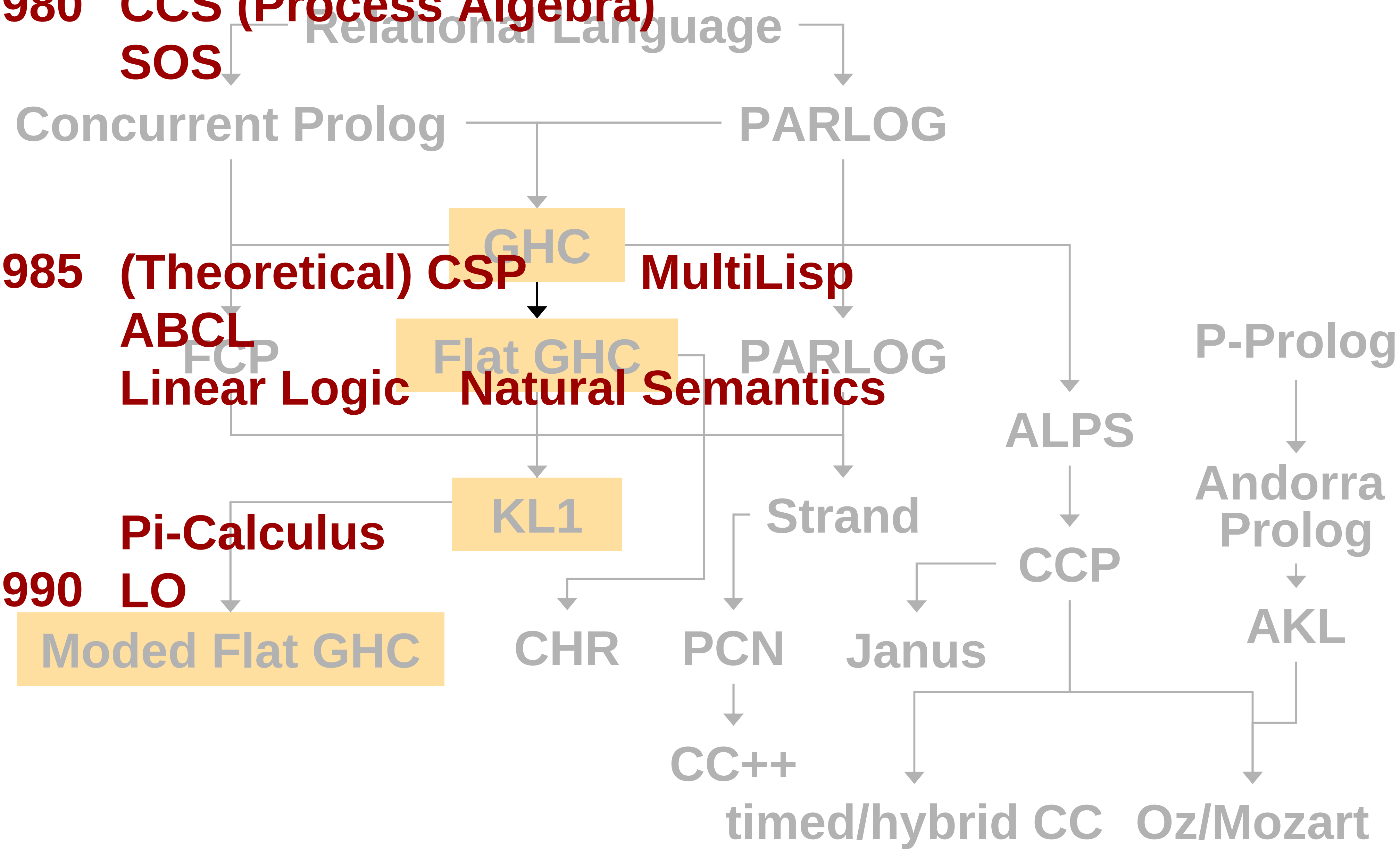
Early History of Constraint-Based Concurrency

1980 CCS (Process Algebra)
SOS

1985 (Theoretical) CSP MultiLisp
ABCL
Linear Logic Natural Semantics

Pi-Calculus

1990 LO



Offspring of Concurrent LP

- **Concurrent Constraint Programming** (late 1980's)
  - Inspired by Constraint Logic Programming
  - Logical view of communication (**Ask** / **Tell**)
  - Generalization of data domains (esp. multisets)
- **CHR (Constraint Handling Rules)** (early 1990's)
  - Allows multisets of goals in rule heads
  - An expressive multiset rewriting language
  - Many applications (esp. constraint solvers)
- **Timed / Hybrid CCP** (early-mid 1990's)
  - Introduced time, defaults, and continuous change
  - High-level language for timed and hybrid systems

Further Readings

- Logic Programming and Concurrency: a Personal Perspective
  - The ALP NewsLetter, Vol. 19, No. 2, 2006 (6 pages).
- Concurrent Logic/Constraint Programming: The Next 10 Years
  - In The Logic Programming Paradigm: A 25-Year Perspective, Springer, 1999, pp. 53-71.
- The Fifth Generation Project: Personal Perspectives
  - Comm. ACM, Vol. 36, No. 3, 1993, pp. 65-76.

Concurrent Logic/Constraint Programming: The Next 10 Years

Kazunori Ueda
Waseda University

Grand Challenges

- **A "λ-calculus" in concurrency field**

  cf. *X*-calculus (calculus of *X*)

  X: π, action, join, gamma, ambient, ...
- **Common platform for non-conventional computation** (parallel, distributed, embedded, real-time, mobile)
- **Type systems** (in the broadest sense) and frameworks of analysis for both logical and physical properties

Two Approaches to Addressing Novel Applications

- **Synthetic**
  - More expressive power
  - Integration of features
- **Analytic**
  - Identifying smaller fragments of LP with nice and useful properties

    cf. Turing machines vs. pushdown automata
  - Separation prior to integration

LP vs. Concurrent LP

- Concurrent LP = LP + choice
  = LP – completeness

???

Choice is essential for specifying arbitration, changes denotational semantics drastically, but otherwise . . .

LP vs. Concurrent LP

- Concurrent LP
  = LP + directionality (of dataflow)
  = Logic
  + embedded concurrency control
- Moded Concurrent LP / CCP:

  ask + tell + strong moding

can/should share more interest with (I)LP

Guarded Horn Clauses and KL1

- Weakest Concurrent Constraint Language
  - ask + eventual tell (asynchronous)
  - parallel composition
  - hiding
  - nondeterministic choice
- A realistic language as well as a model
  - value passing
  - data structures (cf. CCS, CSP, . . .)

Logical Variables as Communication Channels

- Data- and demand-driven communication
- Messages with reply boxes
- First-class channels (encoded as lists or difference lists)
- Replicable read-only data
- Implicit redirection across sites

MEXT 21st Century COE Program (2002-2007)
Ultra-scalable Basic Software Technologies

downward scalability — mm — m — km — **upward scalability**

~~dedicated technologies~~ ~~ad hoc extension~~

embedded and small/nanoscale ← **ordinary scale (PC/WS/clusters) + classical programming models** → **wide-area**

~~ad hoc extension~~ ~~dedicated technologies~~

logical + physical constraints

software design ≈ logical considerations

logical + physical constraints

software technology must address physical constraints

From Turing machine and RAM models to universal models and languages that integrate logic (correctness) and physics (cost)

Basic software must be seamless, simple, verifiable and robust

LMNtal (pronounce: "elemental ")

"Turing machine" for universal computing environments, covering wide-area down to embedded computing

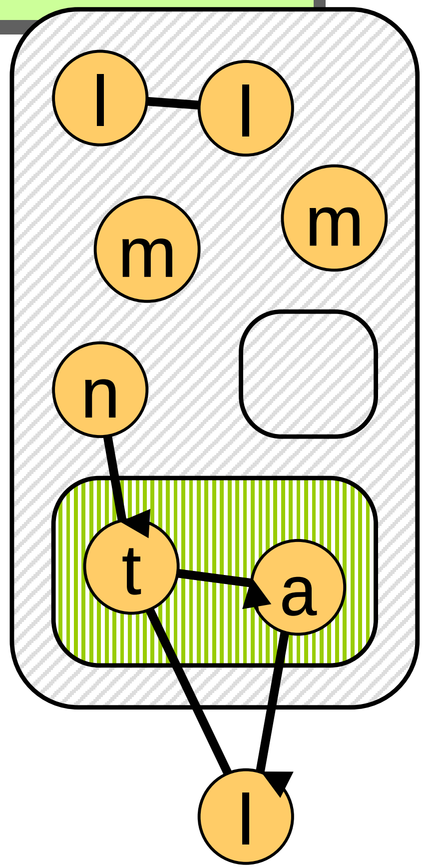
$\mathcal{L}$ = "logical" links

$\mathcal{M}$ = multisets / membranes

$\mathcal{N}$ = (nested) nodes

ta = transformation

$\mathcal{l}$ = language



LMNtal: What and Why

- Rule-based concurrent **language** for expressing & rewriting both **connectivity** and **hierarchy**
  - **Connectivity** and **hierarchy** are the two structuring mechanisms found in many fields ranging from **society** to **biology**, not to mention the world of **computing**
- Computation is manipulation of **diagrams**
  - **Links** express 1-to-1 **connectivity**
  - **Membranes** express **hierarchy** and **locality**
  - Allows **programming by self-organization**

Models and languages with multisets and symmetric join

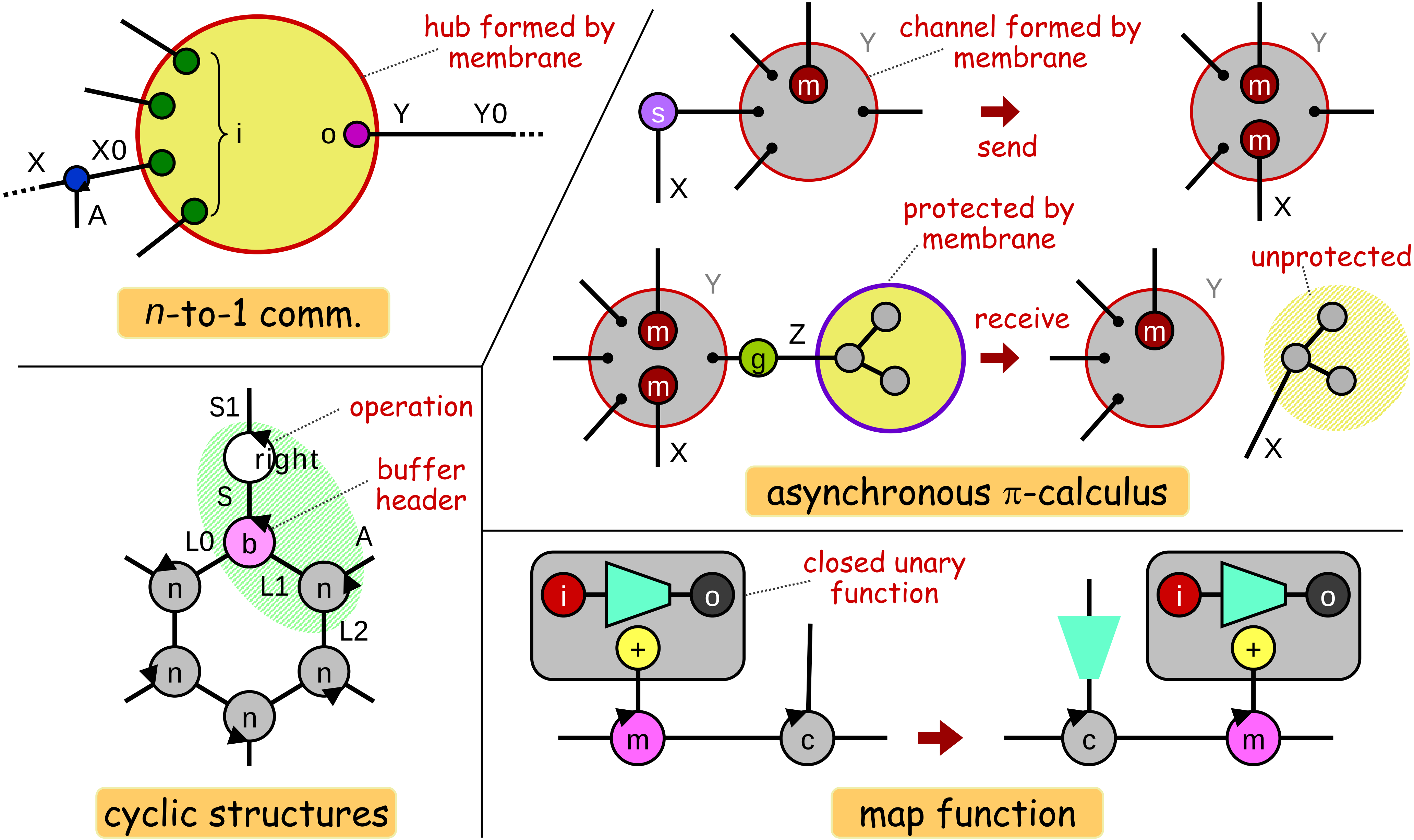
- **Petri Nets**
- **Production Systems** and RETE match
- Graph transformation formalisms
- **CCS, CSP**
- Concurrent logic/constraint programming
- Linda
- **Linear Logic** languages
- Interaction Net
- **Chemical Abst. Machine**, reflexive CHAM, Join Calculus
- Gamma model
- Constraint Handling Rules
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- Bigraphical reactive system

Models and languages with membranes + hierarchies

- **Petri Nets**
- **Production Systems** and RETE match
- Graph transformation formalisms *
- *CCS, CSP*
- Concurrent logic/constraint programming
- Linda *
- Linear Logic languages
- Interaction Net
- **Chemical Abst. Machine**, reflexive CHAM, Join Calculus
- Gamma model
- Constraint Handling Rules
- Mobile ambients
- P-system, membrane computing
- Amorphous computing
- Bigraphical reactive system
- Seal calculus
- Kell calculus
- Brane calculi

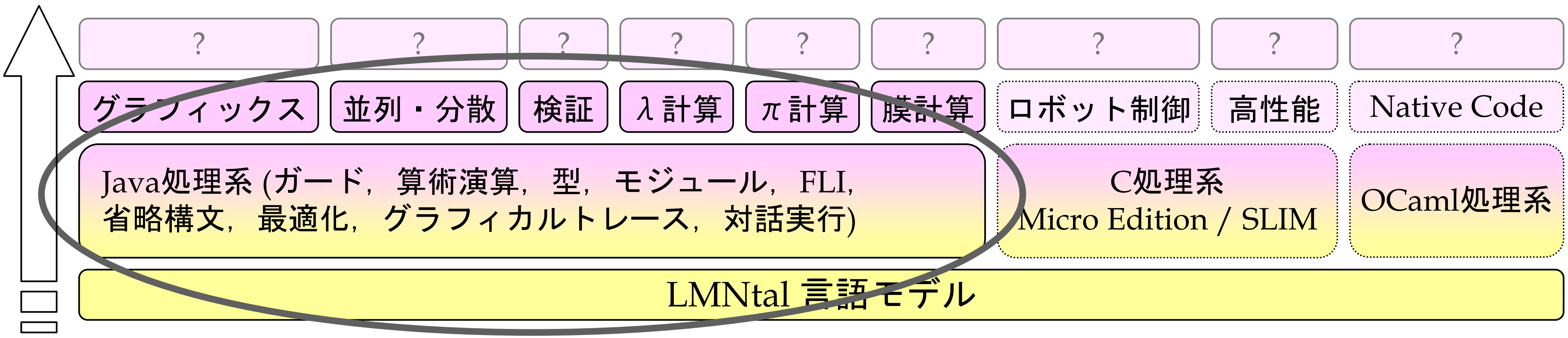
* : some versions feature hierarchies

Expressive power of hierarchical graphs



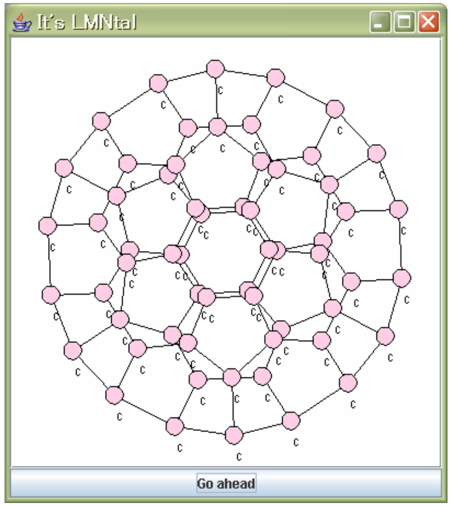
LMNtal: Overview

- Computational **model** based on **hierarchical graph rewriting**
  - Non-directional links (logical variables)
  - Membranes
  - Fine-graind concurrency
- Full-fledged impl. as a practical **language**



LMNtal: Language and implementation

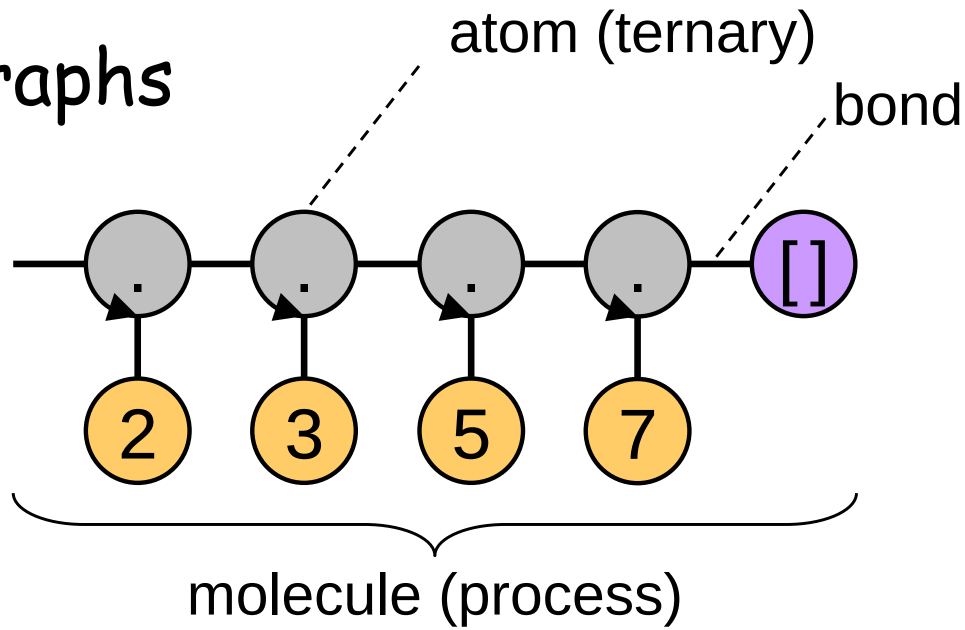
- Language
  - Developed since 2002, tested from many angles
    - K. Ueda and N. Kato, LNCS 3365, etc.
- Implementation
  - Translator to Java running on JDK 1.5
    - **http://www.ueda.info.waseda.ac.jp/lmntal/**
  - 50,000 LOC, very low entry barrier
  - Dedicated intermediate code
  - Features: Module systems / Foreign-language interface to Java / Visualizer / interactive mode / optimizer / library APIs etc.

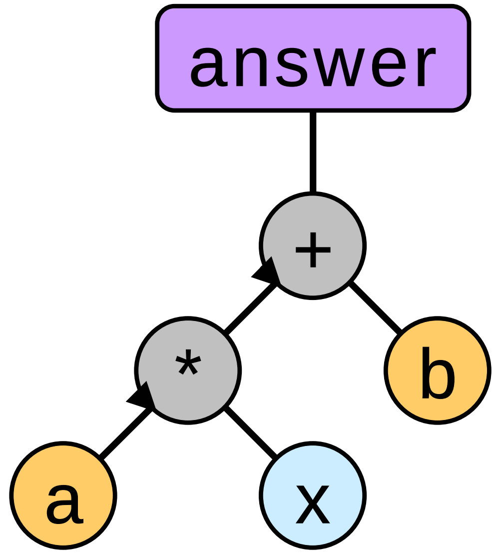


C60 generated from 2 rules and 2 atoms

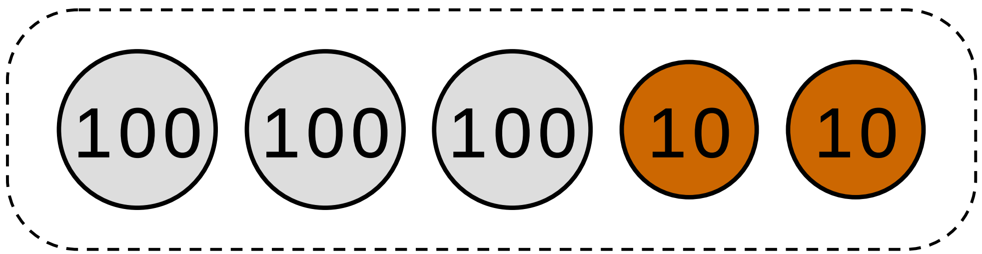
Graphs and Multisets

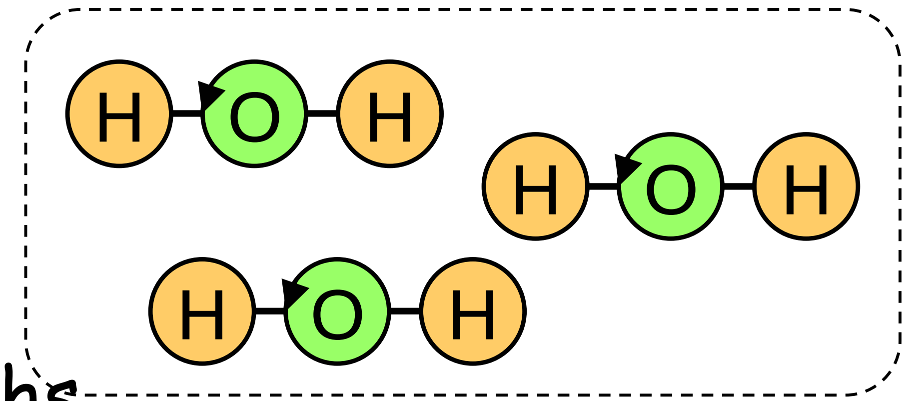
- Graphs





- Multisets



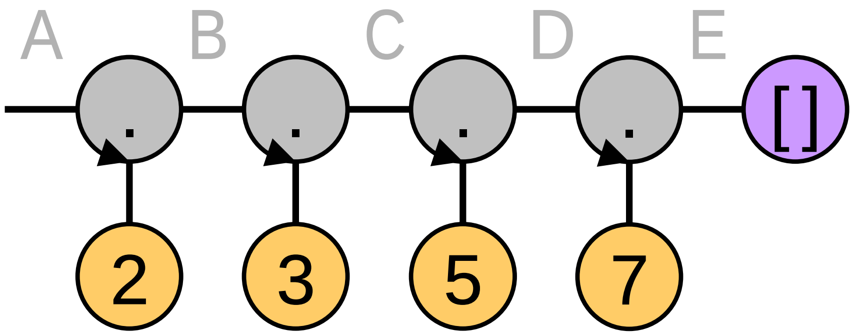


  - Note: multisets are graphs

Text representation

Link (link variables)

- Graphs



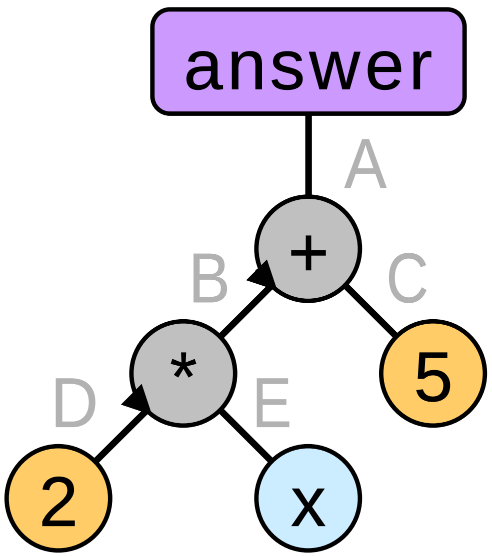
'.'(2,B,A), '.'(3,C,B), '.'(5,D,C), '.'(7,E,D), '[]'(E)

– or –

A= '.'(2,'.'(3,'.'(5,'.'(7,'[]'))))

– or – **A=[2 | [3 | [5 | [7 | []]]]]**

– or – **A=[2,3,5,7]**



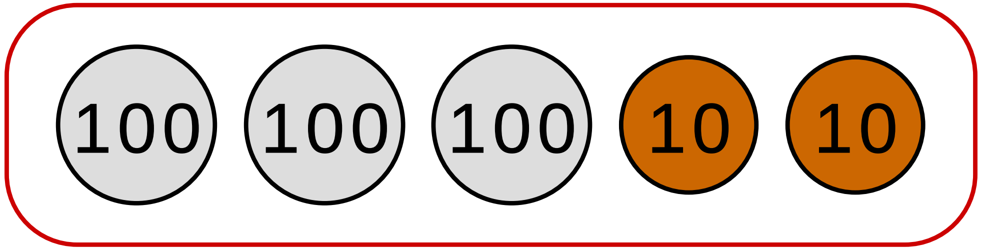
answer(A),
+(B,C,A), *(D,E,B),
2(D), x(E), 5(C)

– or –

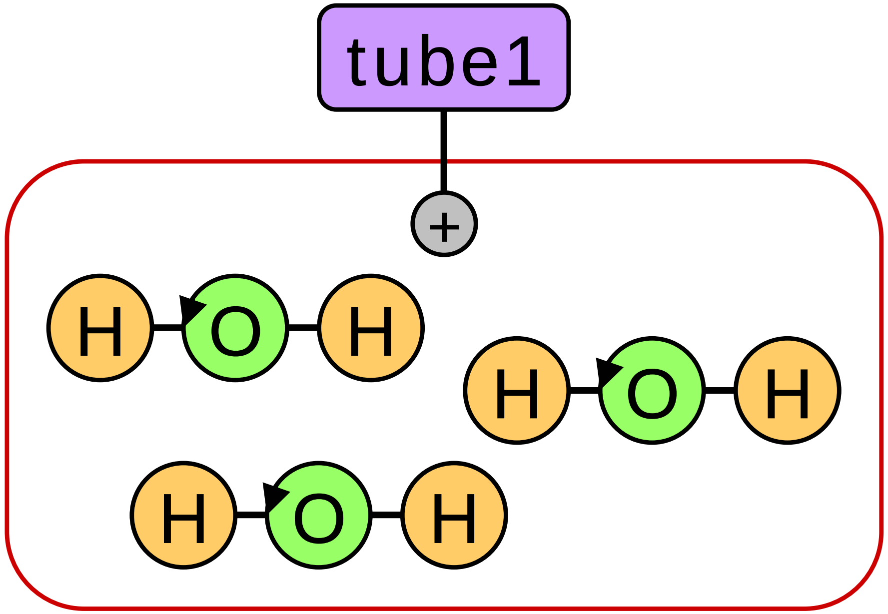
answer(+(*(2,x),5))

Text representation

- Multisets and cells



{ 100, 100, 100, 10, 10 }



tube1(X), { +(X), 'H'(A), 'O'(A,B), 'H'(B), 'H'(C), 'O'(C,D), 'H'(D), 'H'(E), 'O'(E,F), 'H'(F) }

– or –

tube1({ 'O'('H','H'), 'O'('H','H'), 'O'('H','H') })

Pure lambda calculus (1)

graph copying

β-reduction

```
H=apply(lambda(A,B), C) :- H=B, A=C.

lambda(A,B)=cp(C,D,L), {+L,$q} :-
    C=lambda(E,F), D=lambda(G,H), A=cp(E,G,L1), B=cp(F,H,L2),
    {{+L1},+L2,sub(S)}, {super(S),$q}.

apply(A,B)=cp(C,D,L), {+L,$q} :-
    C= apply(E,F), D= apply(G,H), A=cp(E,G,L1), B=cp(F,H,L2),
    {+L1,+L2,$q}.

cp(A,B,L1)=cp(C,D,L2), {{+L1,$p},+L2,$q} :-
    A=C, B=D, {{$p},$q}.

cp(A,B,L1)=cp(C,D,L2), {{+L1,$p},$q}, {+L2,top,$r} :-
    C=cp(E,F,L3), D=cp(G,H,L4), {{+L3,+L4,$p},$q},
    A=cp(E,G,L5), B=cp(F,H,L6), {+L5,+L6,top,$r}.

$u=cp(A,B,L), {+L,$q} :- unary($u) | A=$u, B=$u, {$q}.
```

Pure lambda calculus (2)

graph destruction

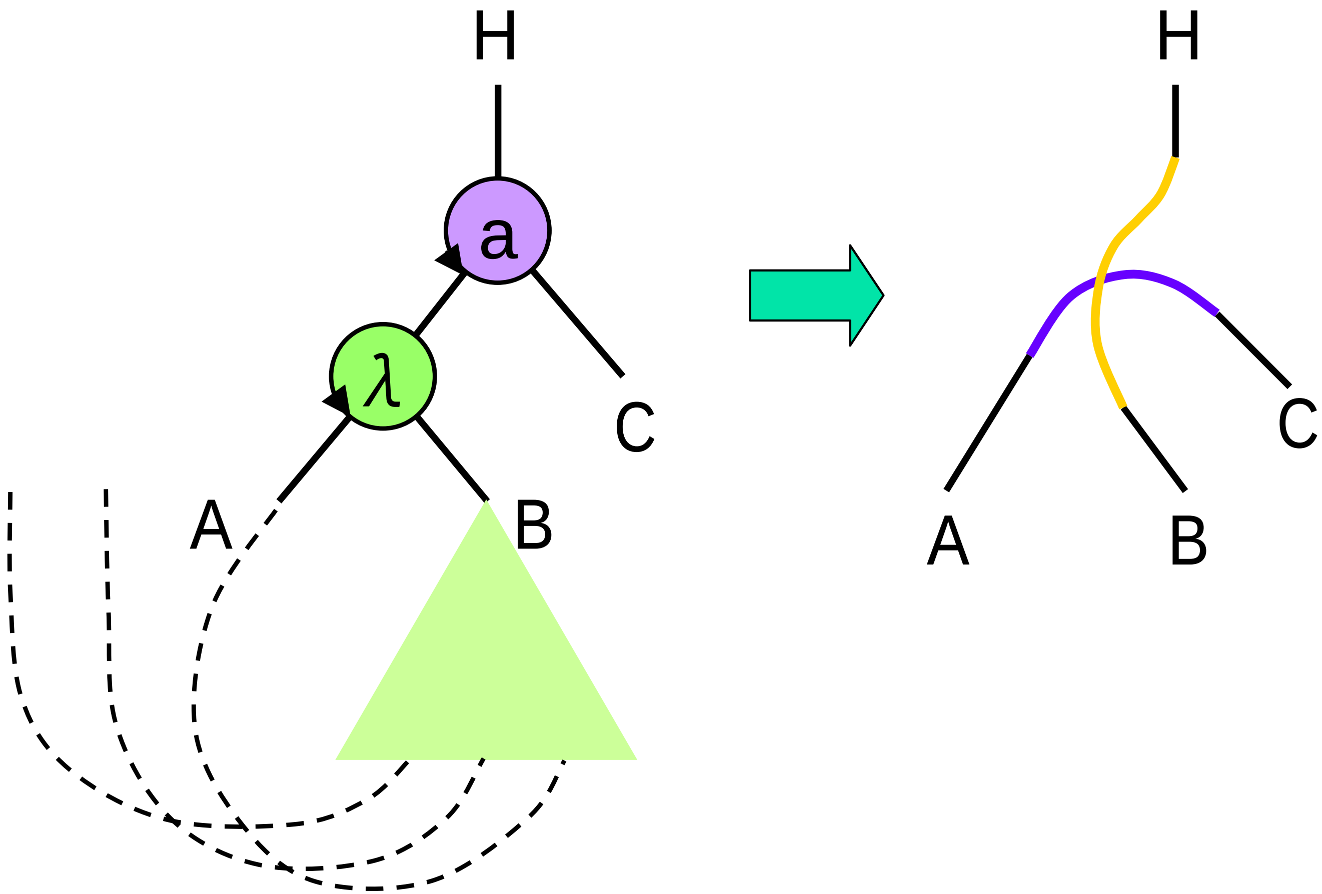
```
lambda(A,B)=rm :- A=rm, B=rm.
apply(A,B)=rm :- A=rm, B=rm.
cp(A,B,L)=rm, {+L,$q} :- A=rm, B=rm, {$q}.
cp(A,B,L)=rm, {{+L,$p},$q} :- A=rm, B=rm, {{$p},$q}.
rm=rm :- .
$u=rm :- unary($u) | .
```

```
{{},$p,sub(S)}, {$q,super(S)} :- {$p,$q}.
A=cp(B,C) :- A=cp(B,C,L), {+L,top}.
{top} :- .
```

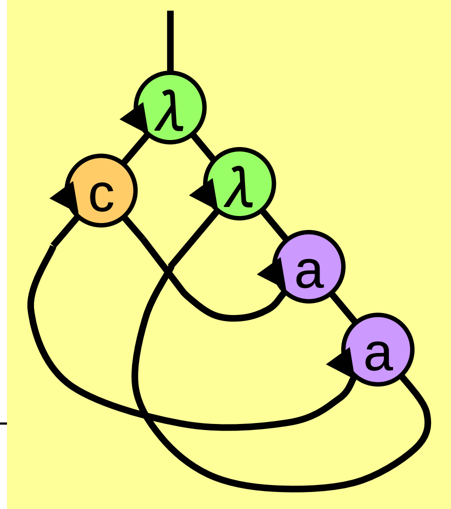
color management

Pure lambda calculus (3)

H=apply(lambda(A,B), C) :- H=B, A=C.



Pure lambda calculus (4)



- Church numeral 2: $\lambda f.\ \lambda x.\ f(f\ x)$

```
lambda(cp(F0,F1),
   lambda(X,apply(F0,apply(F1,X))), Result).
```

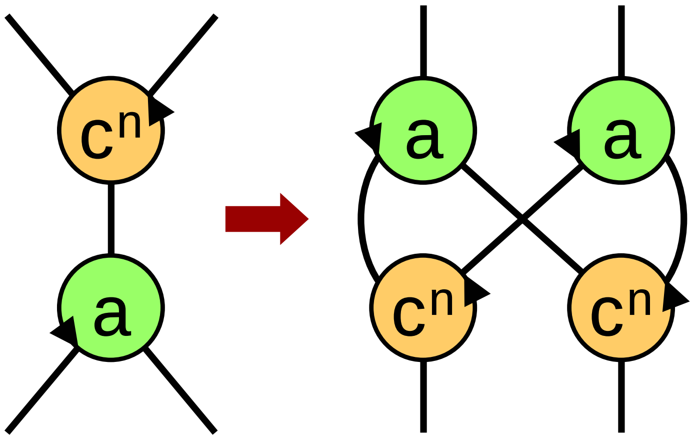
- 3^2 : $(((\lambda m.\lambda n.nm)\ 3)\ 2)$

```
N=n(2) :- N=lambda(cp(F0,F1),
  lambda(X, apply(F0,apply(F1,X)))).
N=n(3) :- N=lambda(cp(F0,cp(F1,F2)),
  lambda(X, apply(F0,apply(F1,apply(F2,X))))).
res=apply(apply(apply(n(2), n(3)), succ), 0).
H=apply(succ, I) :- int(I) | H=I+1.
```

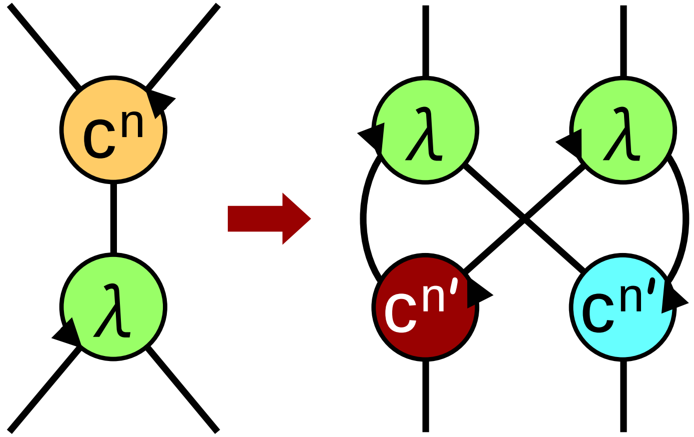
converting to numbers

applying succ and 0 to the Church numeral 3^2

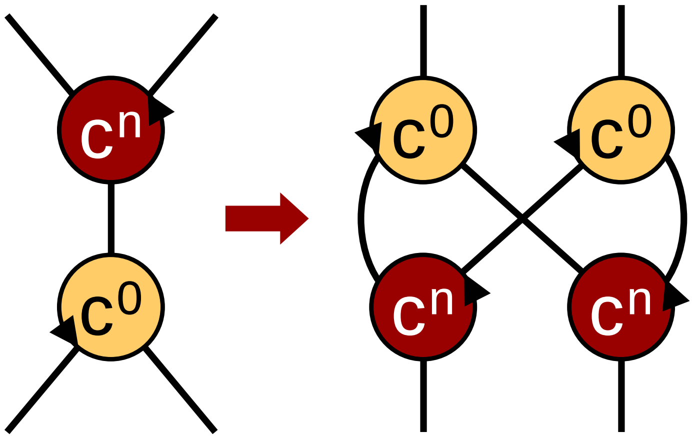
Pure lambda calculus (5)



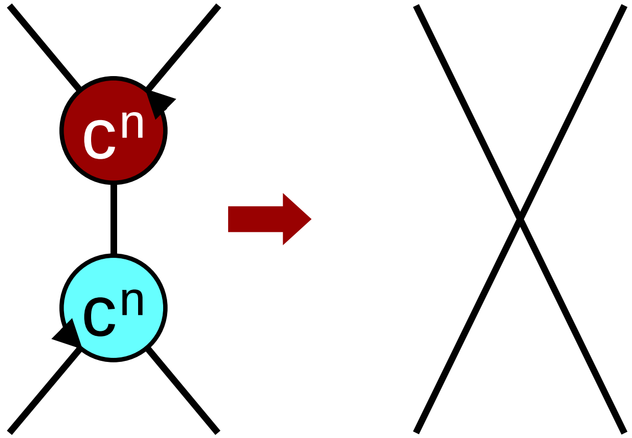
(apply-cp)



(lambda-cp)



(cp-cp1)



(cp-cp2)

Recent development of the LMNtal project

Challenges
- Provably correct and provably efficient parallel software
- Parallel verification

Implementation
- SLIM: high-speed, light-weight backend based on translation to C source (coming soon)

Foundations
- Implementation with guranteed complexity
- Linear logic semantics

New directions
- Engine for model checking