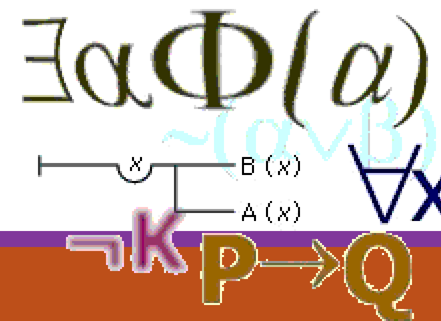
Parallel Execution of Logic Programs: Back to the Future (??)



Enrico Pontelli
Dept. Computer Science
New Mexico State University

Overview

1. Some motivations
   [Logic Programming and Parallelism]
2. The Past
   [Types of Parallelism, Basic Schemes]
3. The Present
   [Recent Schemes]
4. (Back to) The Future
   [New Directions]

MOTIVATIONS AND BASIC DEFINITIONS

Logic Programming

- Definite programs
  - collection of first-order Horn clauses

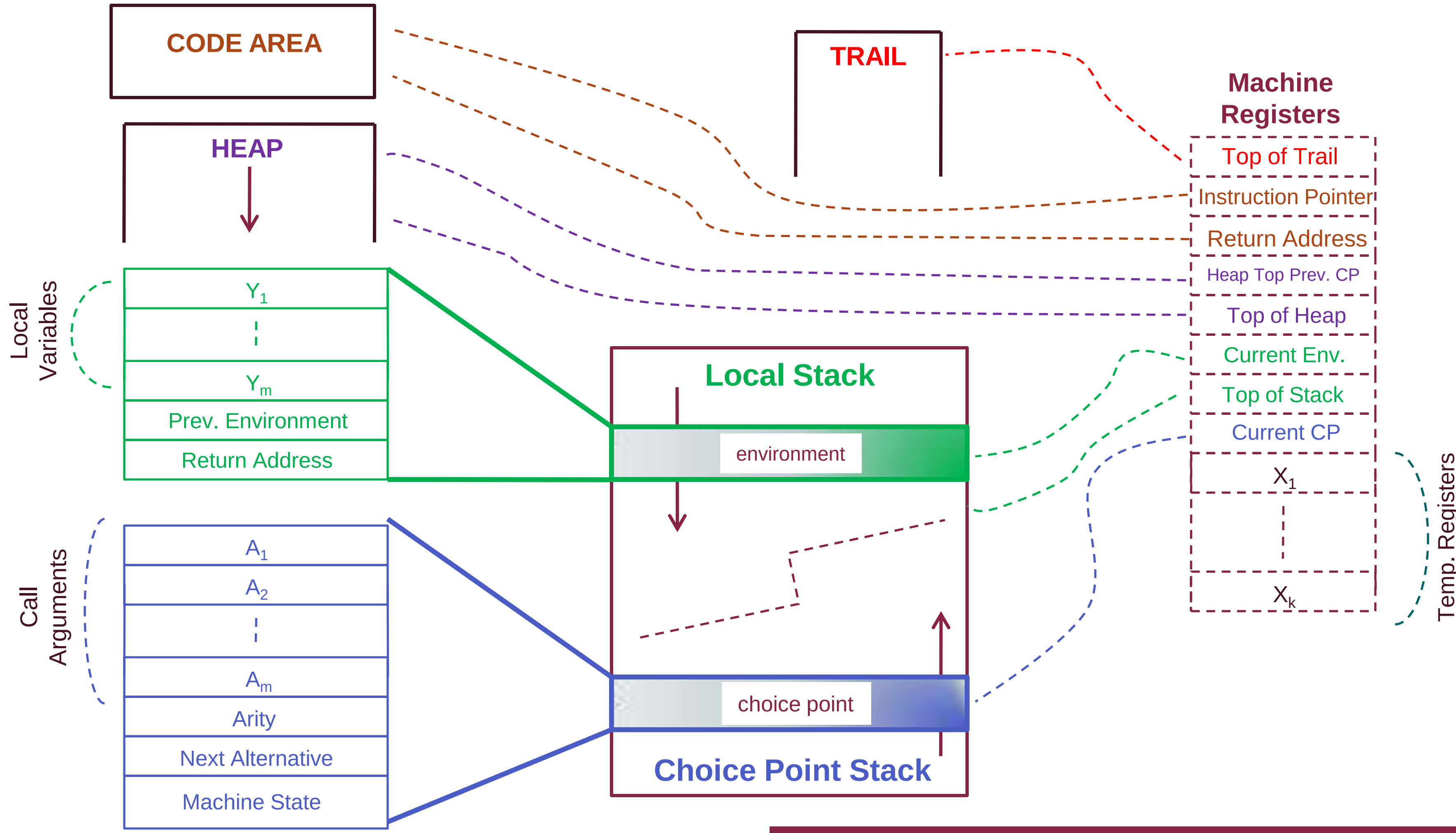
    reachable(X) :- edge(Y,X), reachable(Y).
  - semantics based on least Herbrand model
- Normal programs
  - enter negation as failure

    color(X,red) :- node(X), not color(X,blue).
  - several alternative semantics
    - well-founded semantics [XSB, tabling]
    - answer set semantics [Answer Set Programming]

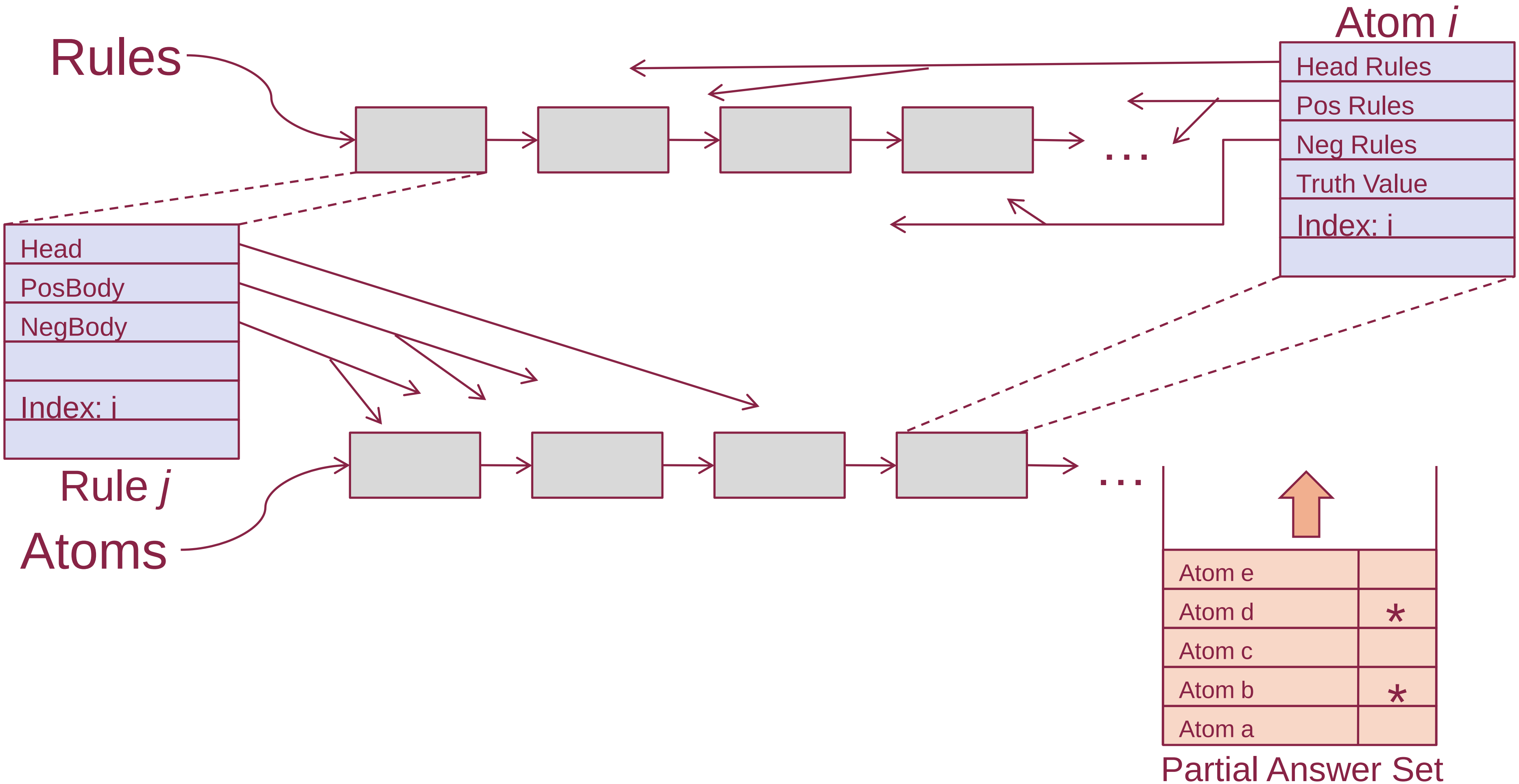
Logic Programming: Systems

- Definite Programs (Prolog, CLP)
  - SLD-resolution
  - WAM-based
- Answer Set Programs
  - bottom-up execution models (answer = set)
    - variations of DPLL
    - mapping to SAT

Logic Programming: WAM



Logic Programming: Smodels



LP and Parallelism

- LP considered suited for parallel execution since its inception
  - Kowalski "Logic for Problem Solving" (1979)
  - Pollard's Ph.D. Thesis (1981)
- Interest spawned by
  - LP $\Rightarrow$ Declarative Language $\Rightarrow$ Limited or No Control $\Rightarrow$ Limited Dependences $\Rightarrow$ Easy Parallelism
  - Everlasting myth of "*LP = slow execution*"

LP and Parallelism

- Several good surveys

J. Chassin de Kergommeaux, P. Codognet.(1994) “Parallel logic programming systems,” ACM Computing Surveys, 26(3).

V. Santos Costa. (2000) “Parallelism and Implementation Technology for Logic Programming Languages,” Encyclopedia of Computer Science and Technology, Vol. 42.

G. Gupta, E. Pontelli, M. Hermenegildo, M. Carlsson, K. Ali. (2001) “Parallel Execution of Logic Programs,” ACM TOPLAS, 23(4).

Overview

1. Some motivations
   [Logic Programming and Parallelism]
2. The Past
   [Types of Parallelism, Basic Schemes]
3. The Present
   [Recent Solutions]
4. (Back to) The Future
   [New Directions]

The Past

- Approaches
  - Explicit Schemes
    - *message passing primitives* (e.g., Delta-Prolog)
    - *blackboard primitives* (e.g., Jinni, CIAO Prolog)
    - *dataflow/guarded languages* (e.g., KLIC)
  - Implicit Schemes
  - Hybrid Schemes

Models of Parallelism

while (Query not empty) **do**
  select$_{literal}$ B from Query — *And-Parallelism*
  repeat
    select$_{clause}$ (H :- Body) from Program — *Or-Parallelism*
  until (unify(H,B) or no clauses left) — *Unification Parallelism*
  if (no clauses left) **then** FAIL
  else
    σ = MostGeneralUnifier(H,B)
    Query = ((Query \ {B}) ∪ Body)σ
  endif
endwhile

Unification Parallelism

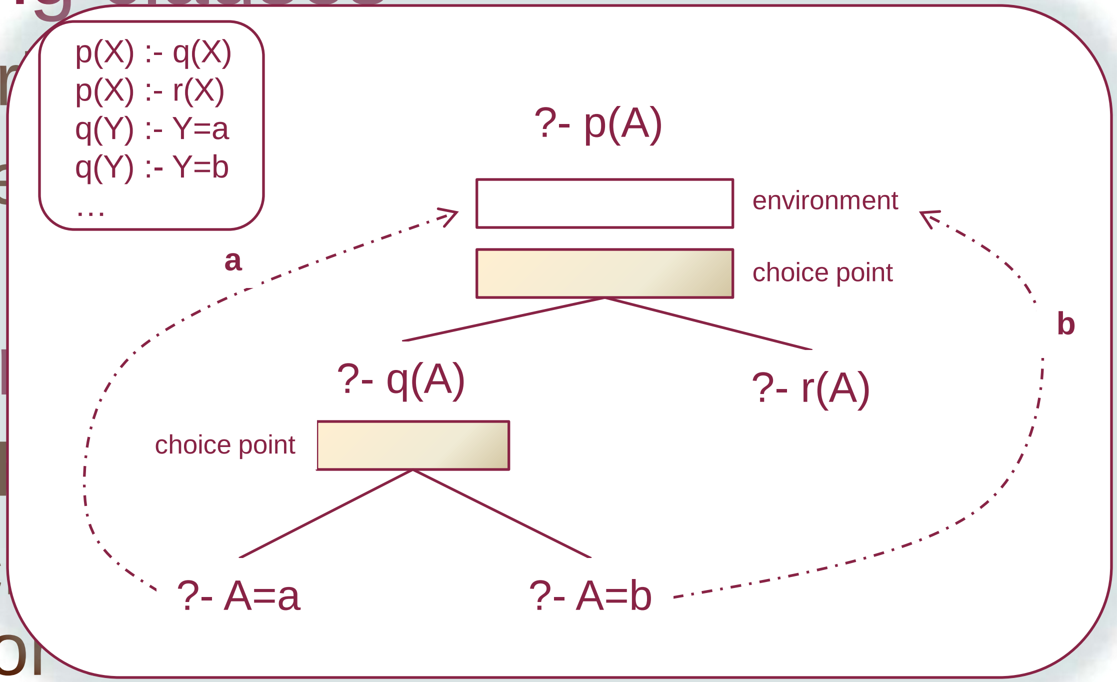
- Parallelize term-reduction stage of unification

$$f(t_1,\ldots,t_n) = f(s_1,\ldots,s_n) \mapsto \begin{pmatrix} t_1 = s_1 \\ \vdots \\ t_n = s_n \end{pmatrix}$$

- Not a major focus
  – fine grained
  – dependences – common variables
  – SIMD algorithms (e.g., Barklund & Millroth)

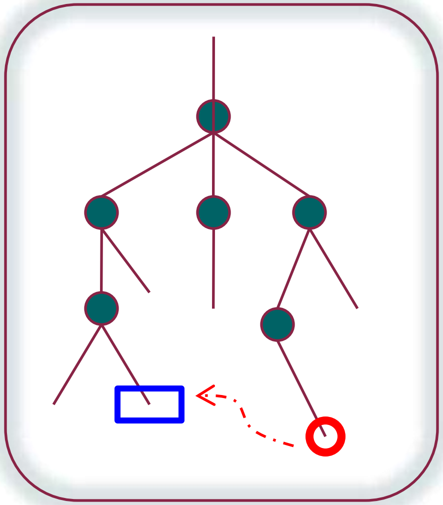
Or-Parallelism

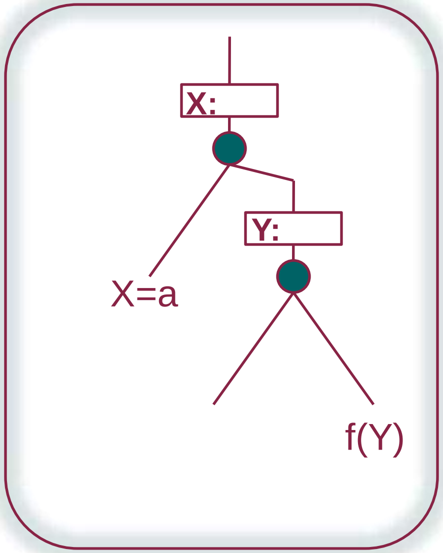
- Parallelize *"don't known"* non-determinism in selecting matching clauses
  - processes explor
  - computations are independent
- Environment repr
  - conditional varial
  - at minimum: eac unbound ancesto



Or-Parallelism: Classification Schemes

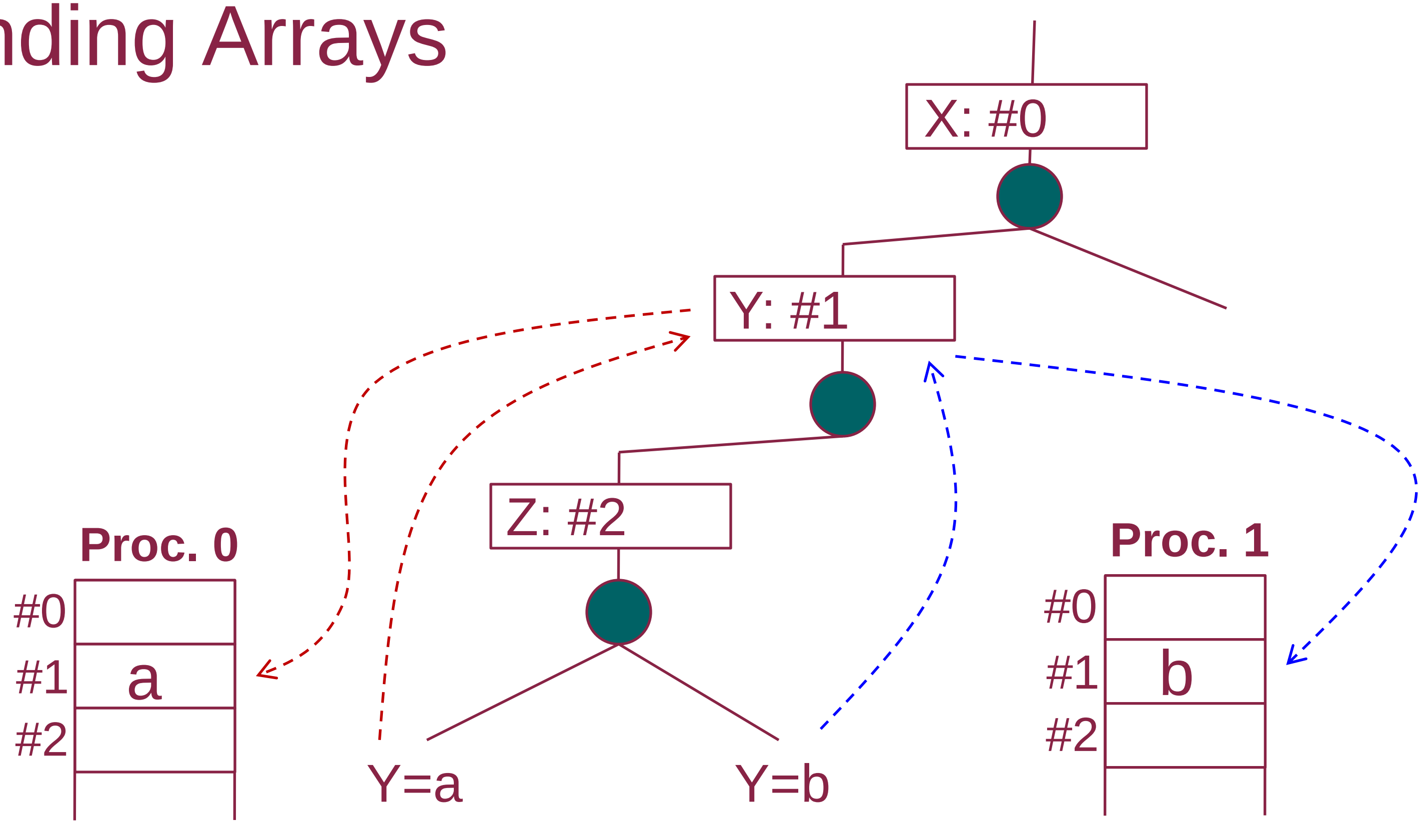
- Formalized in terms of three basic operations
  - binding management scheme
  - task switching scheme
  - task creation scheme
- Binding Scheme
  - Shared-tree methods
    - Binding arrays
    - Version vectors
  - Non shared-tree methods
    - Stack copying
- Task Switching Scheme
  - copying schemes
  - recomputation schemes





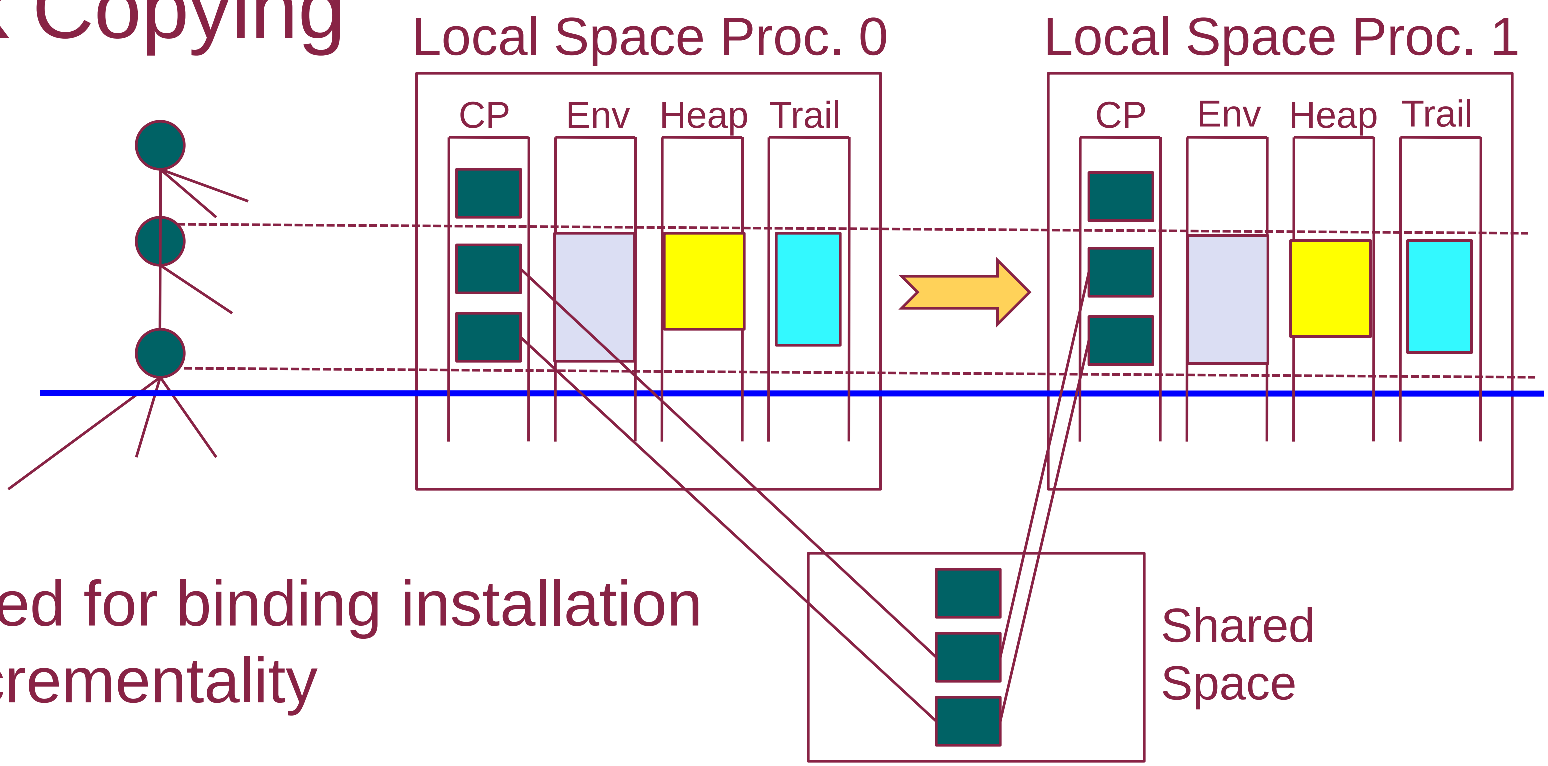
Or-Parallelism: two popular schemes

- Binding Arrays



Or-Parallelism: two popular schemes

- Stack Copying



- need for binding installation
- incrementality

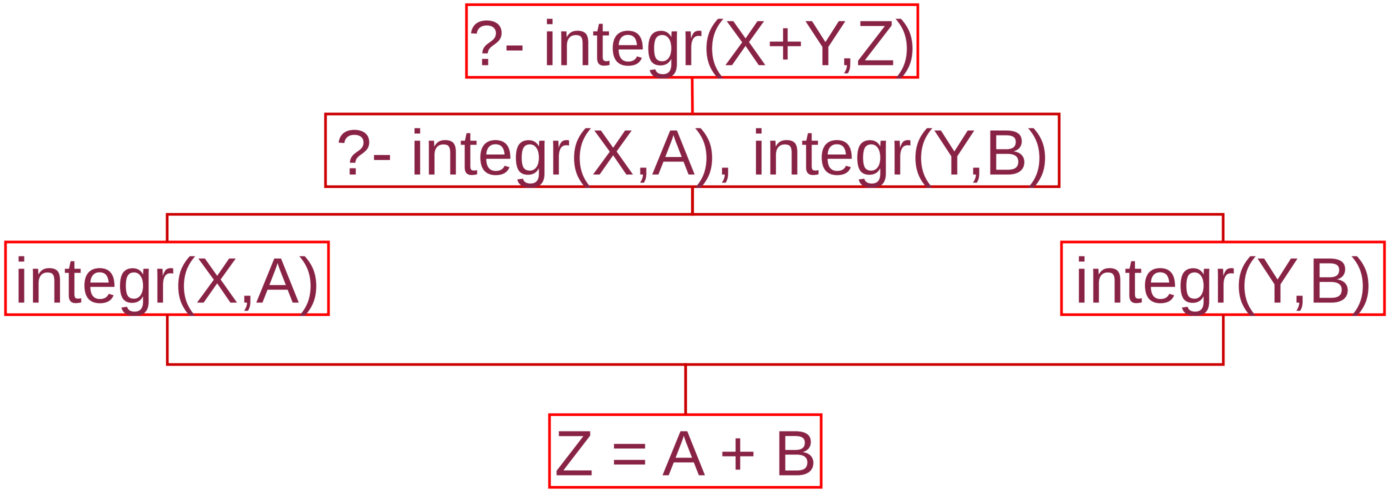
And-Parallelism

- Concurrent execution of different literals in a resolvent

integr(X + Y, Z) :- integr(X,A), integr(Y,B), Z = A + B.

parallel literals: integr(X,A), integr(Y,B)

continuation: Z = A + B.



And-Parallelism

- Two traditional forms
  - Independent and-parallelism
    - runtime access to independent sets of variables

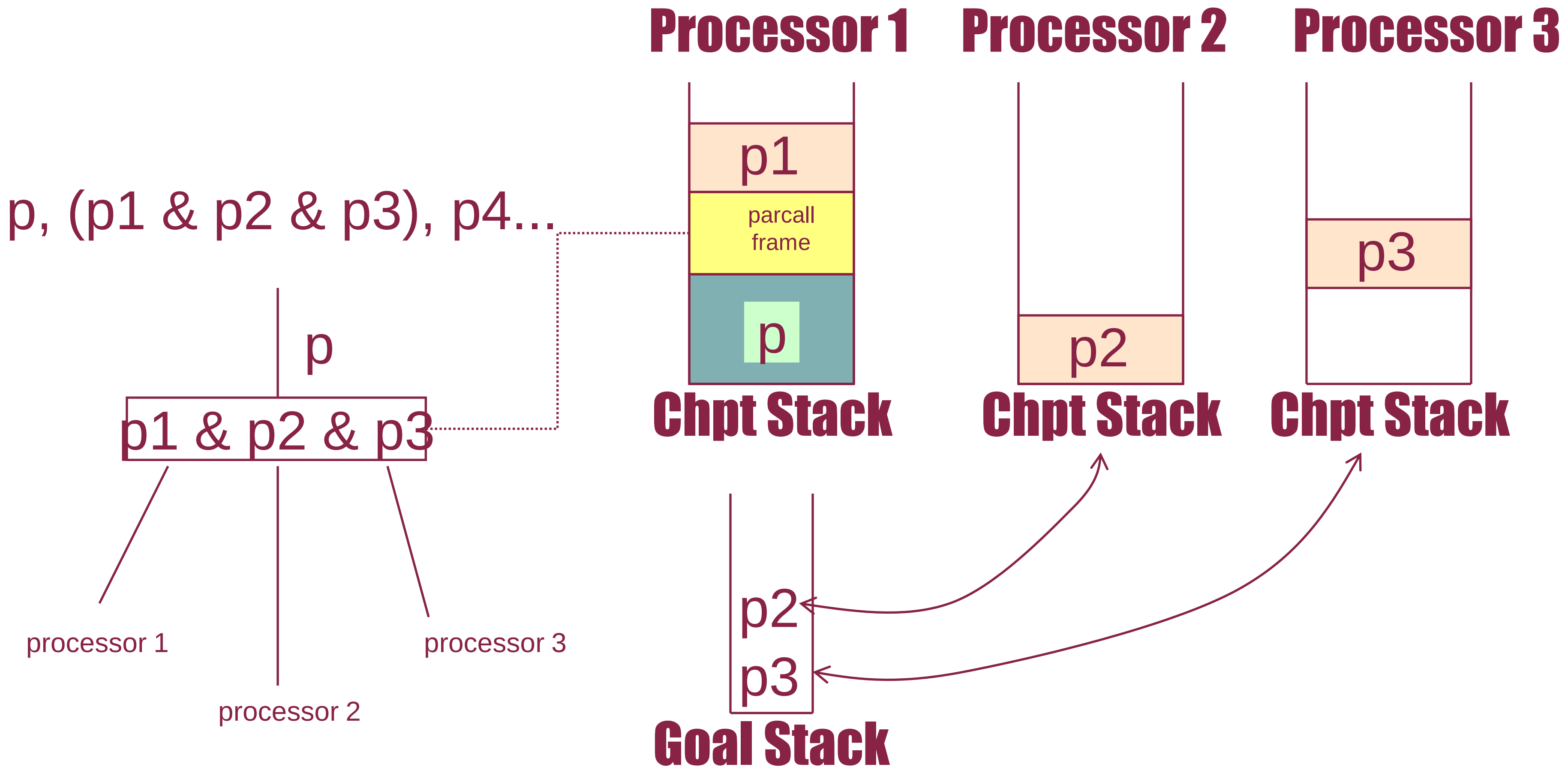
```
quick(In,Out) :- partition(In,First,Low,High),
                 ( indep(Low,High) => quick(Low,SLow) & quick(High,SHigh)
                   ;    quick(Low,SLow) , quick(High,SHigh)
                 ),
                 append(SLow,[First|SHigh],Out).
```

test (indep(Low,High) =>)

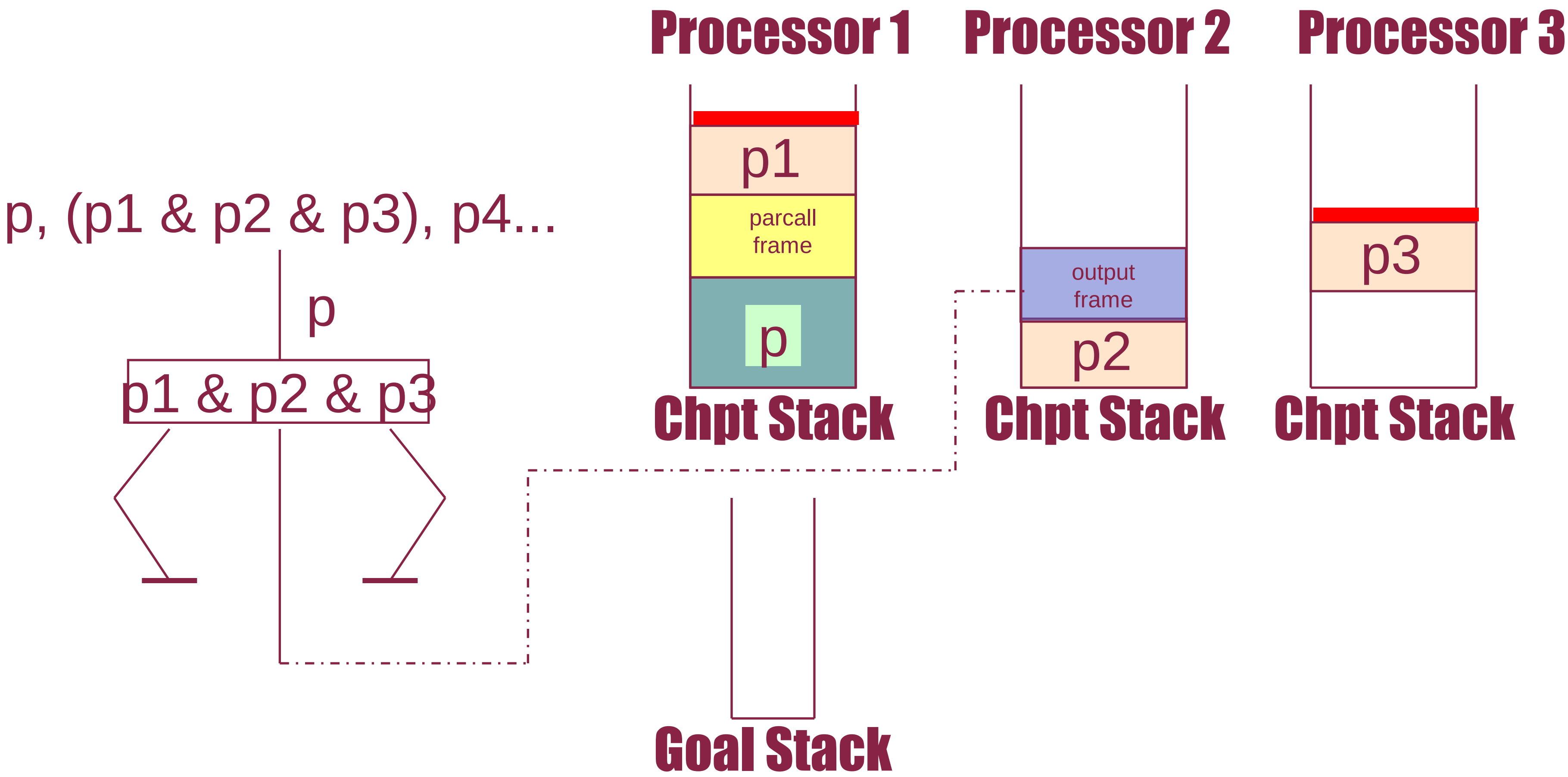
parallel case (quick(Low,SLow) & quick(High,SHigh))

sequential case (quick(Low,SLow) , quick(High,SHigh))

And-Parallelism



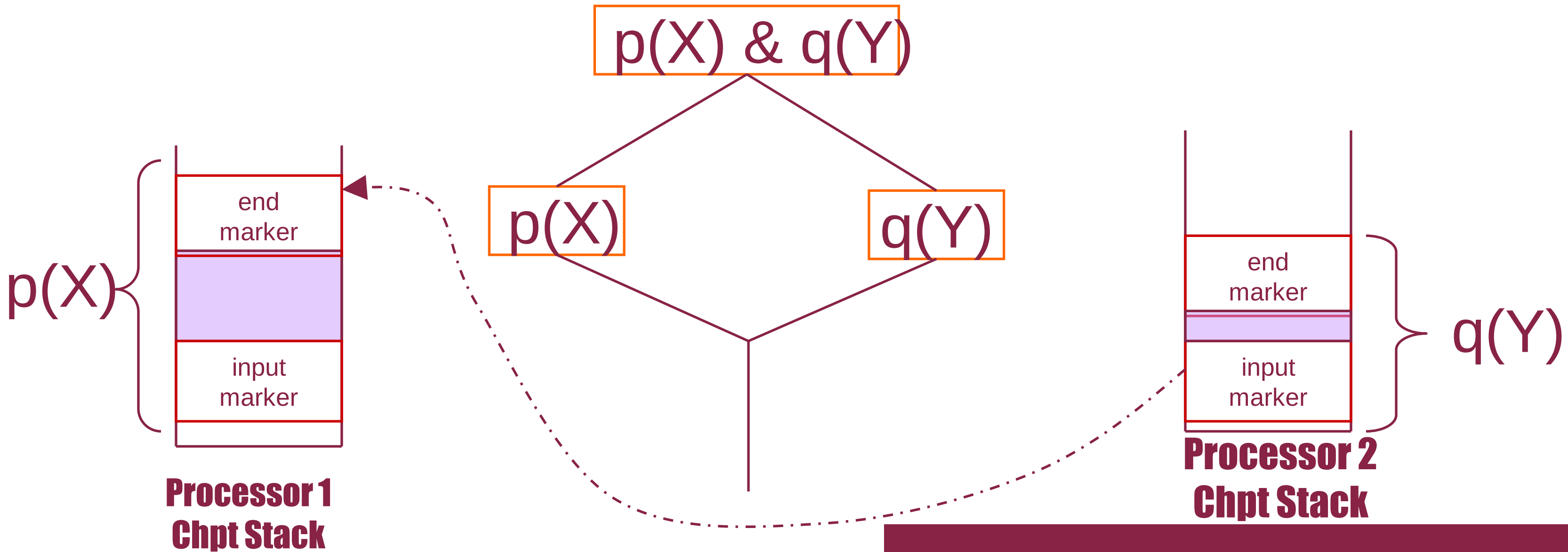
And-Parallelism



And-Parallelism

- Backtracking

p1(..), (<cond> ⇒ p2(..) & p3(..) & p4(..)), p5(..)



And-Parallelism

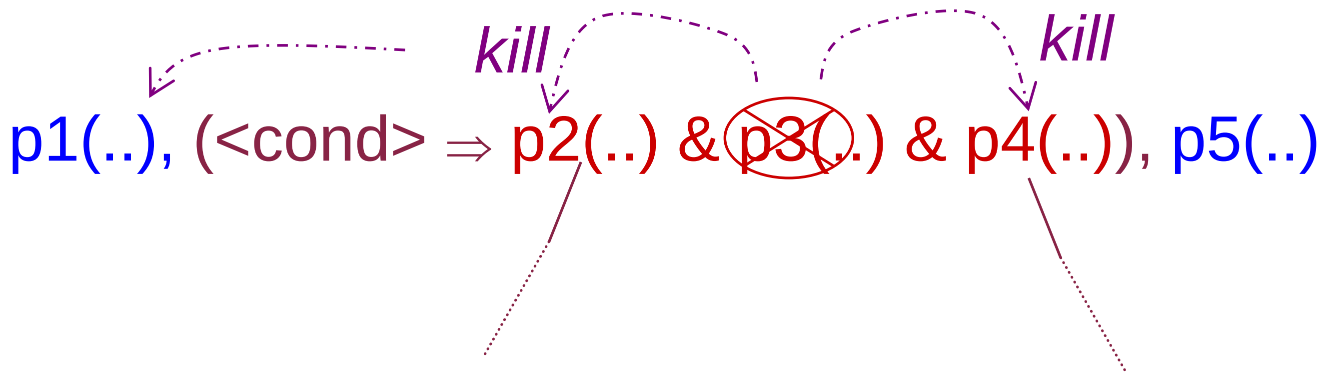
- Outside backtracking

  p1(..), (<cond> $\Rightarrow$ p2(..) & p3(..) & p4(..)), p5(..)

- Standard right-to-left
  - across processors
  - skip deterministic goals
  - restart in parallel

And-Parallelism

- Inside backtracking



And-Parallelism

- Dependent and-parallelism

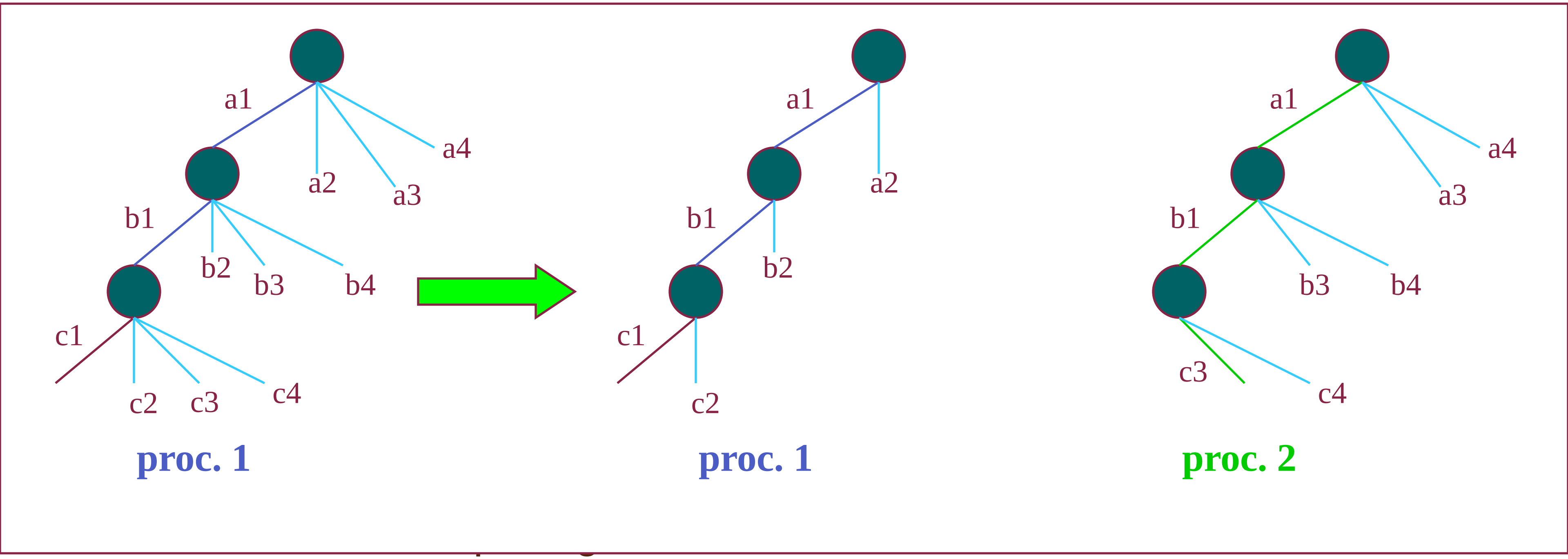
$$p(X) \ \& \ q(X)$$

  - Goals
    - consistent bindings
    - reproduce Prolog observable behavior
  - Common approach
    - dynamic classification of subgoals as producers/consumers
    - several complex schemes (e.g., filtered binding model, DDAS)
  - Complex backtracking

Overview

1. Some motivations
   [Logic Programming and Parallelism]
2. The Past
   [Types of Parallelism, Basic Schemes]
3. The Present
   [Recent Solutions]
4. (Back to) The Future
   [New Directions]

The Present (or recent past...)



- vertical splitting
  - partition: CP* $\rightarrow$ CP* $\times$ CP* e.g.,
    - » alternate($a_1 b_1 a_2 b_2 a_3 b_3 \ldots$) = ($a_1 a_2 a_3 \ldots$),($b_1 b_2 b_3 \ldots$)
    - » block($a_1 a_2 a_3 \ldots a_n$) = ($a_1 \ldots a_{n/2-1}$),($a_{n/2} \ldots a_n$)

Understanding the Problems...

- Formalizations
  - modeling key aspects of parallel LP as problems on dynamic trees
  - investigation of computational properties
- Some interesting results
  - environment representation problem
    - operations: create_tree, expand, remove, assign, dereference
    - $\Omega(\lg n)$

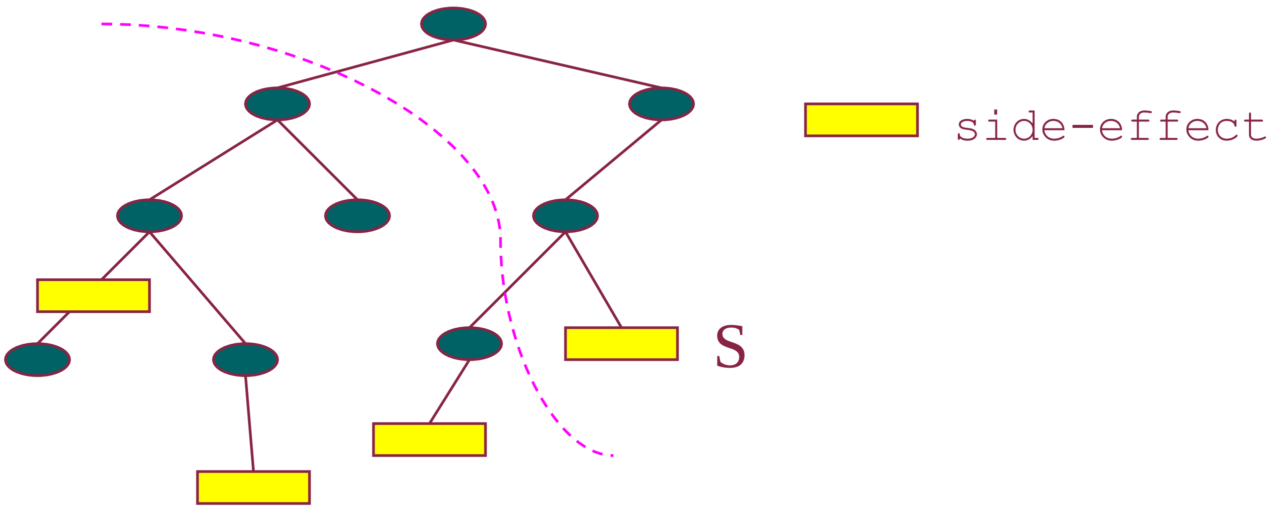
Further Issues

- Prolog as a "real" programming language
  - Side-effects, order-sensitive predicates

  - Goal: recreate the same observable behavior of sequential Prolog
  - Sequentialize order-sensitive predicates
  - Sequential is opposite of Parallel…
  - Dynamic vs. Static management of order-sensitive predicates

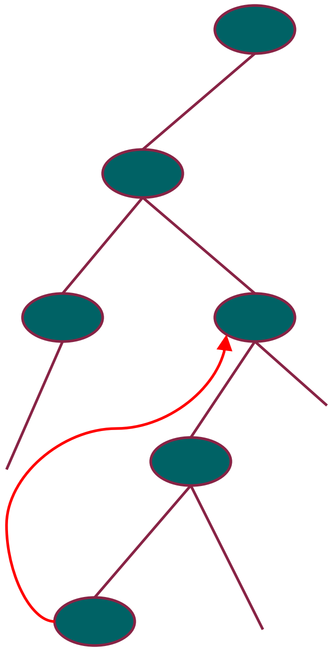
Order-sensitive Executions

- Idea: a side-effect should be delayed until all “preceding” side-effects have been completed
- determining the exact time of execution: undecidable
- safe approximation: delay until all “impure” branches on the left of the side-effects have been completed



Order-sensitive predicates

- Standard Technique: maintain subroot nodes for each node
- Subroot(X) = root of largest subtree containing X in which X is leftmost



- Aurora, Muse: O(n) algorithms for maintaining subroot nodes
- possible to perform O(1) on shared memory
- approximated on distributed memory: block splitting, give away top part of branch
- Preemptive scheduling

Overview

1. Some motivations
   [Logic Programming and Parallelism]
2. The Past
   [Types of Parallelism, Basic Schemes]
3. The Present
   [Recent Solutions]
4. The Future
   [New Directions]

(Back to) The Future

To Japanese Page for "The Home page of KLIC"

This page describes KLIC, which is a concurrent logic programming language.

KLIC was developed in ICOT and distributed as an IFS, and is now distributed by KLIC Association.

Please read the README file for details

- Current version
- Available platforms
- Parallel implementations
- Changes
- Documents
- Mailing lists

Current version

Version 3.003 is the latest and is currently distributed from the following sites:

- http://www.klic.org/files/v3.0/klic-3.003.tgz(primary site)
- ftp://pwd.chroot.org/pub/klic/v3.0/klic-3.003.tgz(secondary site)

This distribution contains the following three implementations.

- Sequential (pseudo parallel) implementation
- Distributed memory parallel implementation
- Shared memory parallel implementation

Unfortunately, the current shared memory parallel implementation still has fatal bugs. If you are using the old shared memory parallel implementation (previous version is 2.002), *please do not replace it with the current version.*

type Status report

message /~beaumont/andorra.html

description The requested resource (/~beaumont/andorra.html) is not available.

Apache Tomcat/5.5.9

type Status report

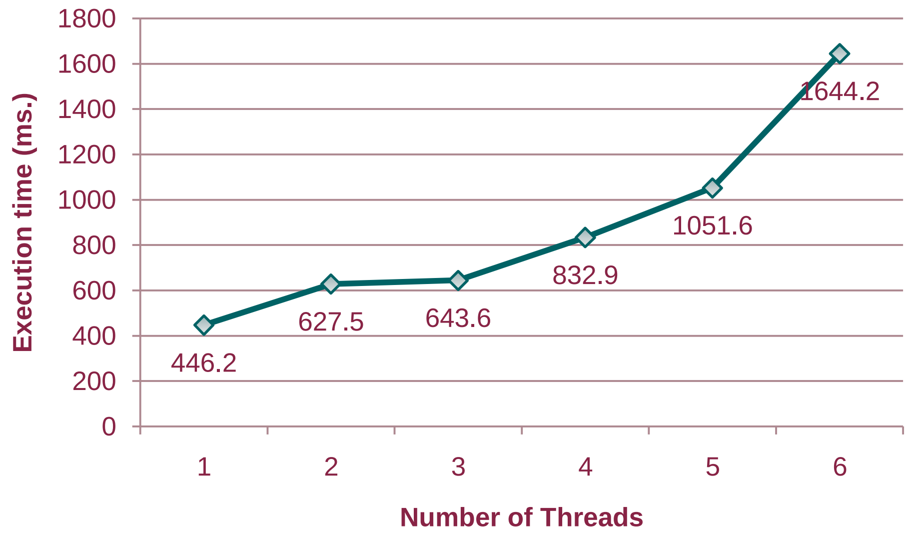
message /~beaumont/aurora.html

description The requested resource (/~beaumont/aurora.html) is not available.

Apache Tomcat/5.5.9

(Back to) The Future… NOT!

- Multi-core
  - back to shared memory platforms
  - small/medium/large scale
  - the future is not the same as the past…
    - GPUs: large number of simp complex memory model
    - CPUs: tricky cache behavio
    - Other upcoming platforms (
- Requirements
  - better investigation of loca
  - hybrid architectures ➲ hyb
    - ACE, Andorra-I, FIRE, AOW



Perspectives

- Mapping forms of parallelism to hw levels
  - Originally designed in ACE
    - Teams of processors
    - Each team as an or-parallel agent (stack copying)
    - Each member of a team as an and-parallel agent (&ACE)
  - Experimented with in Jsmodels
    1. Propagation Parallelism – threads within a multicore node
    2. Or-Parallelism – processes allocated to distinct nodes of a cluster
    - Pathways 10
  - Other possibilities
    - GPUs for unification parallelism

(Back to) The Future… NOT!

- Making parallel LP boring
  - High level parallel primitives
  - Implement forms of parallelism in Prolog
  - Original idea
    - Codish & Shapiro (1987)
    - &-Prolog CGE (1988)
    - &-ACE DAP (1995)
- An Example: primitives for and-parallelism
  - G&>Handler (post goal in a goal queue)
  - Handler &<   (wait for goal associated to Handler)
    - A&B :- A&>Handler, call(B),   Handler &<.
  - Operations for accessing goal list
  - Mutex on variables

(Back to) The Future… NOT!

- Parallelism in
  - Parallel Answ
    - Jsmodels
    - Platypus
  - Jsmodels
    - sequential m
      semantics ar

function jsmodels(P)

    $S = (\emptyset,\emptyset)$

    loop

        $S = wfm(P,S)$

        if $(S^{+} \cap S^{-} \neq \emptyset)$ then fail

        if (S is complete) then return S

        pick f and choose

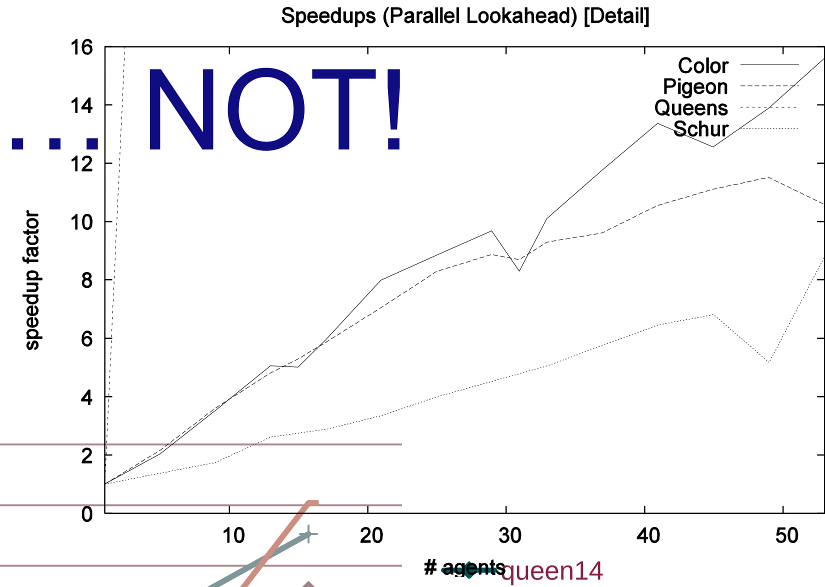
            $S^{+} = S^{+} \cup \{f\}$

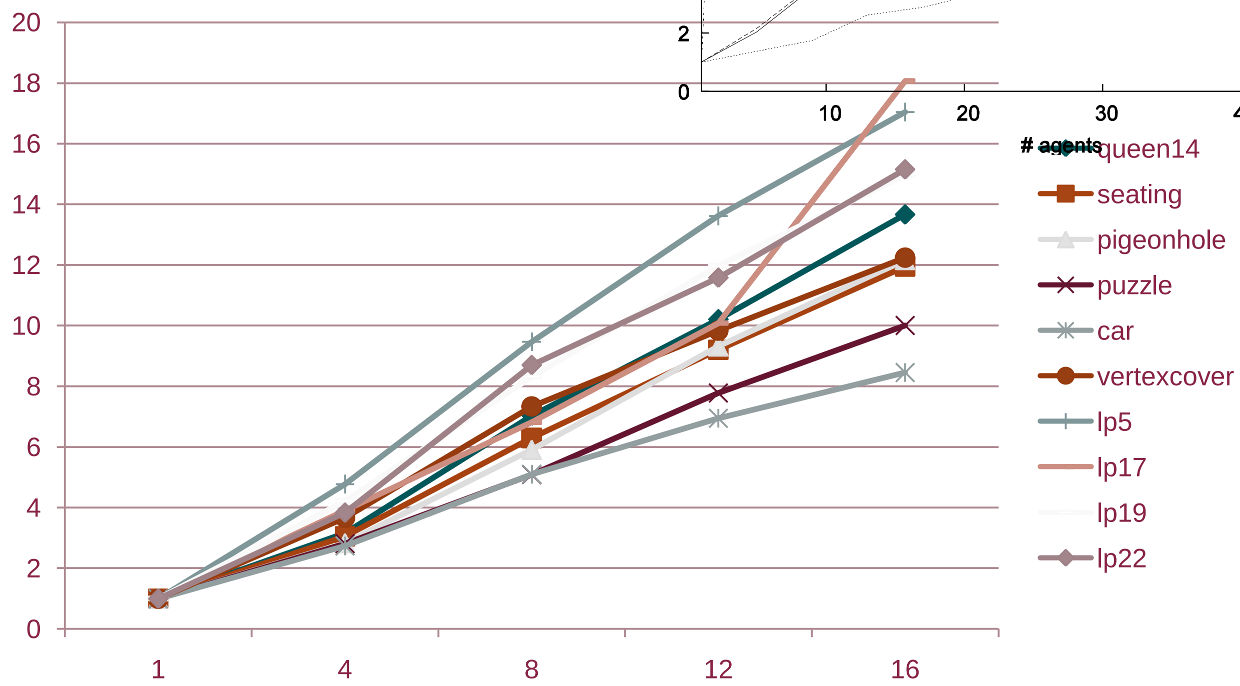
            $S^{-} = S^{-} \cup \{f\}$

    endloop

(Back to) The Future… NOT!

- Parallel jsmodels
  - search pa
    - parallel
    - environ
    - 3 sharir
      - copy
      - reco
      - reco
    - both se
  - lookahea
    - find unk inconsis
    - lookahe parallel



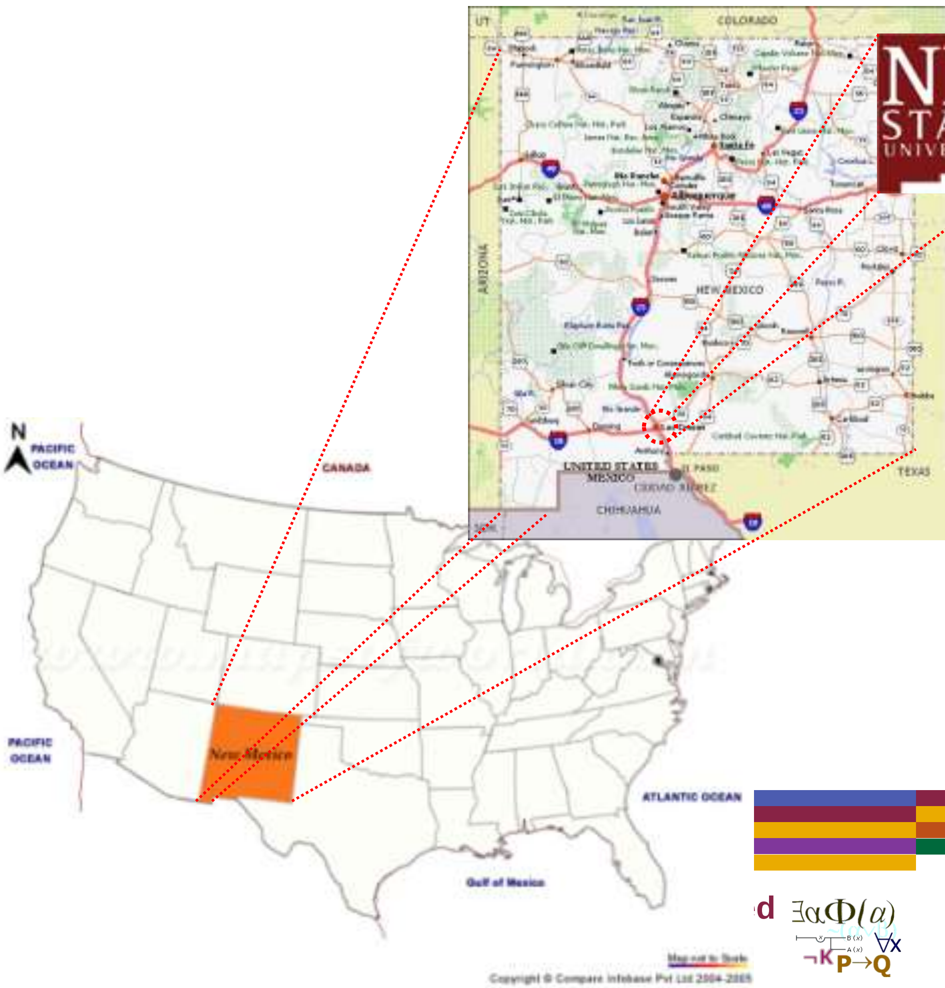


Conclusion

- Parallel LP is coming back
- Novel perspectives
  - architectural impact
  - portability & mainteinability at the price of greater overheads
  - language specific schemes
- Bright future ahead!

Acknowledgments

- KLAP = Knowledge representation, Logic, and Advanced Programming



G. Gupta
U.T.Dallas

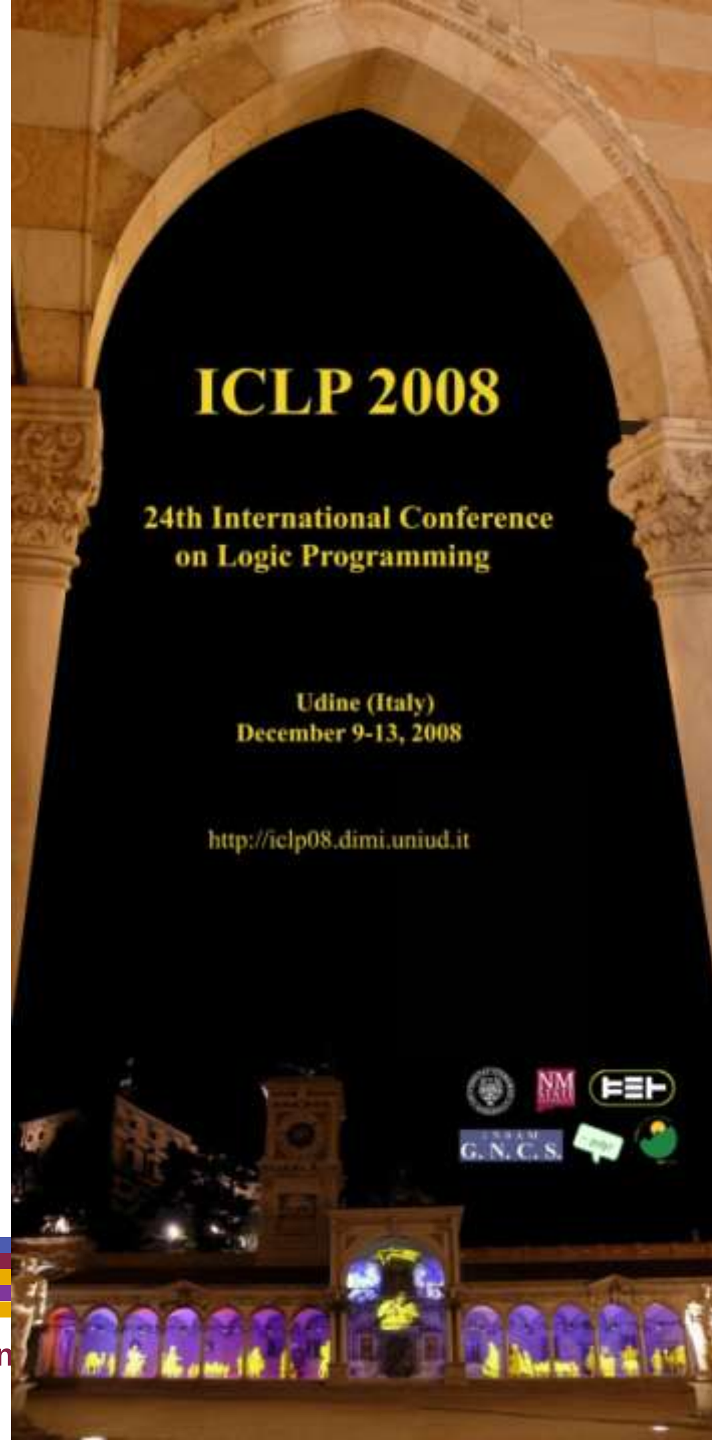
M. Hermenegildo
U.Politecnica Madrid

H. Viet Le
Iowa State U.

K. Villaverde
New Mexico St. U.

M. Carro
U.Politecnica Madrid

V. Santos Costa
U. Porto


ICLP 2008
24th International Conference
on Logic Programming
Udine (Italy)
December 9-13, 2008
http://iclp08.dimi.uniud.it

Thank You

Questions?