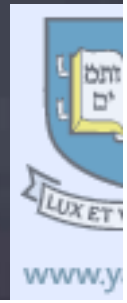
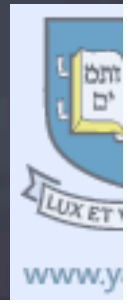
Expressivity

- Drew McDermott
- Yale University



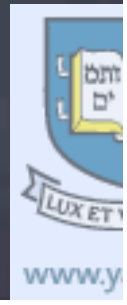
Planning Is...

- Reasoning about alternative courses of action in order to achieve a goal and/or optimize some utility function
- Actions must not interact "too much." (Rubik's Cube is not a planning problem.)



Some History

- Strips vindicated GPS, and gave us addlists & deletelists
- It didn't work all that well. (Control structure problems, search-space problems.)
- "Search needs to be controlled." (Conventional wisdom, ca. 1975)



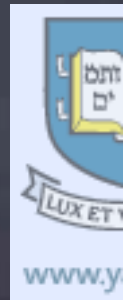
Expressivity $\stackrel{?}{=}$ Control

- Abstrips --> NOAH & Nonlin
  - Abstrips inferred (well, almost...) abstractions of actions (less important goals postponed)
  - NOAH used hierarchical plans with abstractions captured by the person that wrote them

The Dark Ages

Renaissance

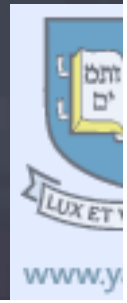
- TWEAK (Chapman), SNLP (McAllester & Rosenblitt)
- Emphasis on provable properties ("systematicity"), which tended toward minimalism
- Prodigy (CMU, Carbonell, Minton, Veloso, Knoblock, ...) --- Strips rekindled



The Carboniferous

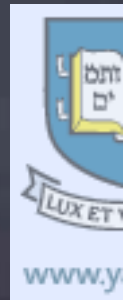
(to make a hash of my temporal metaphors)

- UCPOP (U. Washington, Weld et al. ad inf.) conquers the world, or at least makes it safe for partial-order causal-link (POCL) planning
- Pednault's ADL (Action Description Language) is the standard for elegance and expressivity
  - Secondary preconditions (`when P Q`)



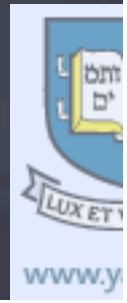
Good Algorithms!

- Graphplan (Blum & Furst), SATPlan (Kautz & Selman), estimated ("relaxed") regression Planning (McDermott; Geffner & Bonet), many, many more, . . .
- Too many to count, let alone survey
- Expressivity took a huge hit. Back to Strips! Any progress on expressivity had to preserve algorithmic efficiency.



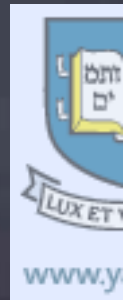
IPC and PDDL

- One key purpose of the International Planning Competition is to push the field forward by increasing expressivity "a bit."
- IPC-1: PDDL (Planning Domain Definition Language)
- IPC-2: No change (except hierarchical notation essentially abandoned)



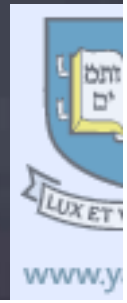
IPC and PDDL

- IPC-3: Durative actions, fluent functions, objective functions
- IPC-4: Timed initial literals & derived predicates



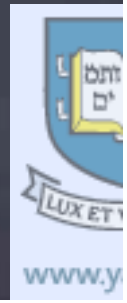
Lessons Learned

- Most flaws in PDDL are due to the temptation to take current algorithms' limitations seriously
  - `:length` field in problem statements (IPC-1)
  - `:functions` must be numerical fluents
  - "Durative actions" (as opposed to autonomous processes)



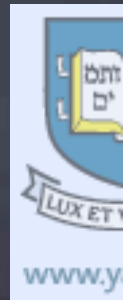
Lessons Learned

- Don't get too far ahead of what we know how to handle.
- Prime example: General-purpose hierarchical plan notation.



Where Do We Go from Here?

- Add HTN planning to PDDL (again)
- Because: Algorithms (e.g., SHOP2) exist. Some problems require canned plans as heuristic guidance.
- If all planners used the same notation, we could compare them (for crying out loud).



SHOP2 Notation

Primitive and compound **tasks**

Primitives are defined by "operators" = (action definitions)

Compounds are defined by **methods**

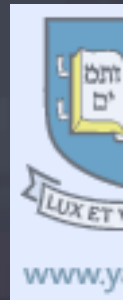
```
(:ordered [method | task ]*)

(:unordered [method | task ]+)
```

The Need for "Dataflow"

- Actions (and processes) should have values as well as effects.
- For knowledge effects, but also to create names for new things

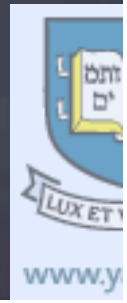
E.g., "correlation tokens"

```
(send ?a ?msg) produces id
-->  used by (receive ?id)
```



SHOP2 & Dataflow

- Uses `assert` and `erase` to store signals in the world model
- Special devices to hide these "actions" in the output
- Blech



Golden et al., U. Washington (mid 90s)

"Run-time" variables:

`(p !a ?b)` -- `'a'` gets set at run time, typically to something one has discovered

This work was (almost) the last word on knowledge conditions, but it didn't address the link issue.

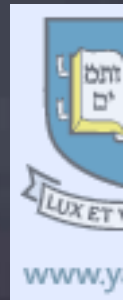
OWL-S Dataflow

```
tagA :: perform A(x <= 5);
...;
perform B(z <= tagA.outY)
```

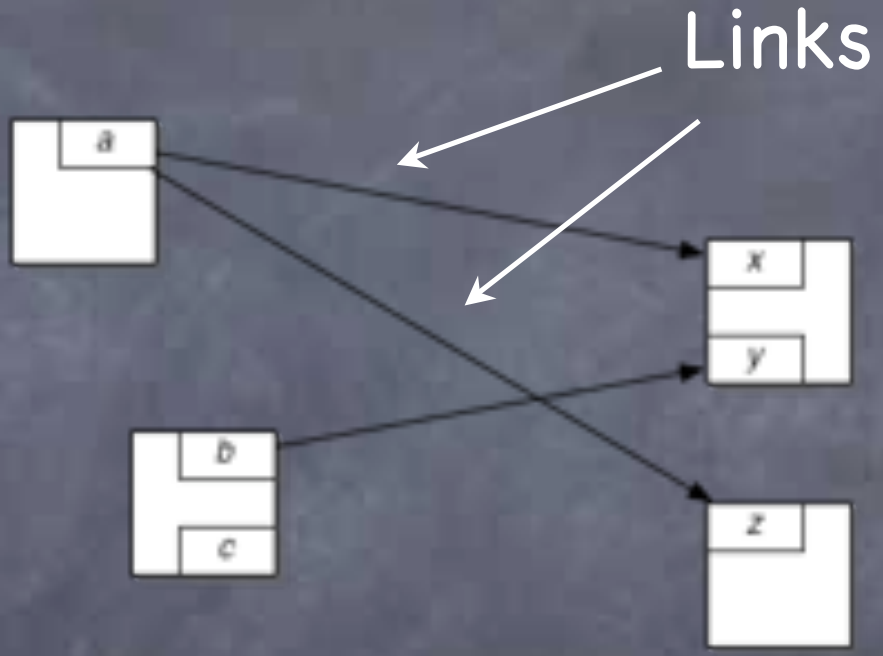
(For those familiar with
RDF/XML OWL-S, this
the "presentation syntax

Not felicitous
given
conditionals...

```
if test then t :: perform A
        else f :: perform B;
perform C(x <= t.z, x <= f.u)
```



Dataflow Between Tasks



(From Williamsson, Decker, & Sycara 1996 "Unified Information and Control Flow in HTNs," AAAI Conf.)

Naming Links

```
(with-links (n - (Val x y))
   (seq (if test (link (perform (A ...))
                       :output n)
                 (link (perform (B ...))
                       :output n))
        (link :input n
              (perform (C (!_y n))))))
```

Wherever possible, try to infer links;
and/or make them look like `:vars` vars

Example

E-commerce notations:

(know-val= *term literal*) - Predicate

(transfer-amt *pred agt1 agt2*)

- Total amount of objects satisfying *pred* transferred from *agt1* to *agt2*

(delta *quant x*) - The change in *quant* (from the initial situation) is *x*

Currency-trading domain:

Goal: (delta (transfer-amt
(currency euro) me Soros)
100000)

Standard Action

```
(:action (transfer ?n1 ?pred ?a)
:effect (increase (transfer-amt ?pred me ?a))
:expansion :methods)
```

Standard Plan

```
(:method (transfer ?n1 (currency ?curr1) ?a)
 :vars (n1 n2 curr1 curr2 a middleper exch)
 :precondition
   (know-val= (price ?middleper (currency ?curr1)
                     ?curr2)
              ?exch
 :expansion
   (seq (trade ?middleper
               (* ?exch ?n1) (currency ?curr2)
               ?n1 (currency ?curr1))
        (transfer ?n1 (currency ?curr1) ?a)))
```

Is HTN Planning Ready to Stretch?

We'll see...

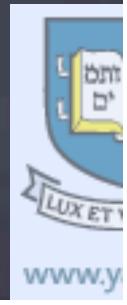
Dealing with dataflow

Contingencies and loops

Unifying HTN & situation-space planning

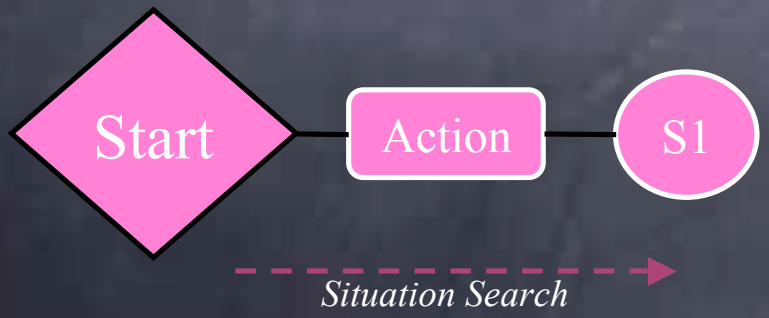
As Expressivity Grows...

- Do as much reasoning as possible "forward," in current situation (not during regression)
- You can "plug in" modules if you have a complete situation representation. Plug-ins might include information-retrieval planners, constraint solvers, schedulers, ...



Estimated-Regression Planning

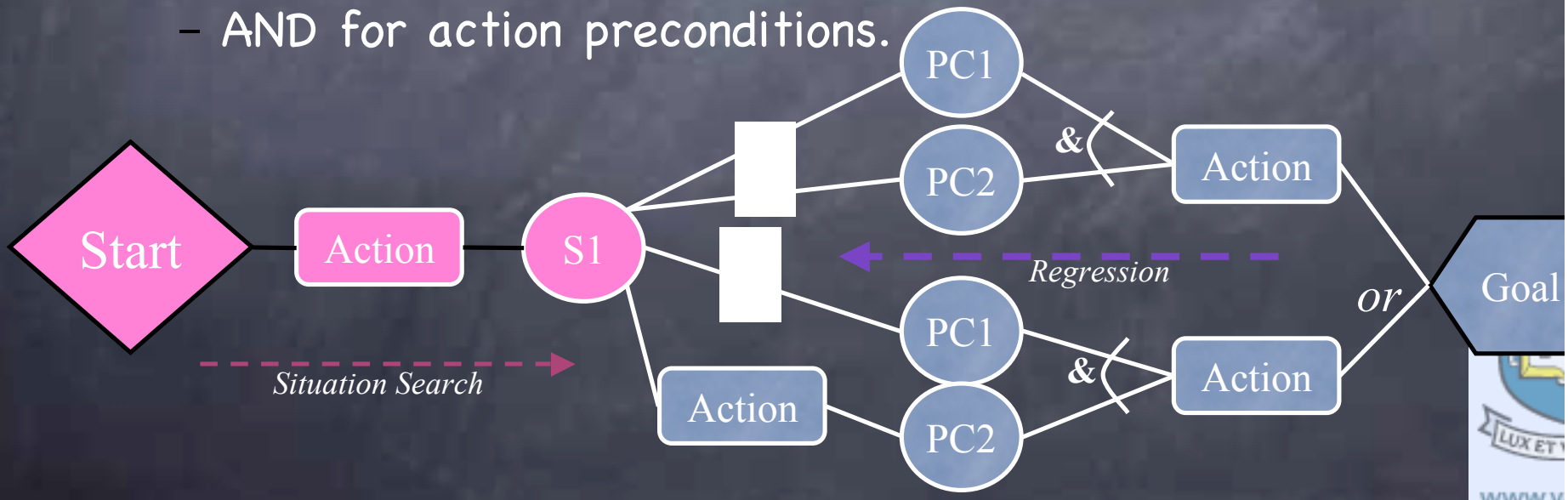
- Situation-space search (= space of plan prefixes)
- Reserve the word "state" for "search state" (= plan prefix + other stuff...)



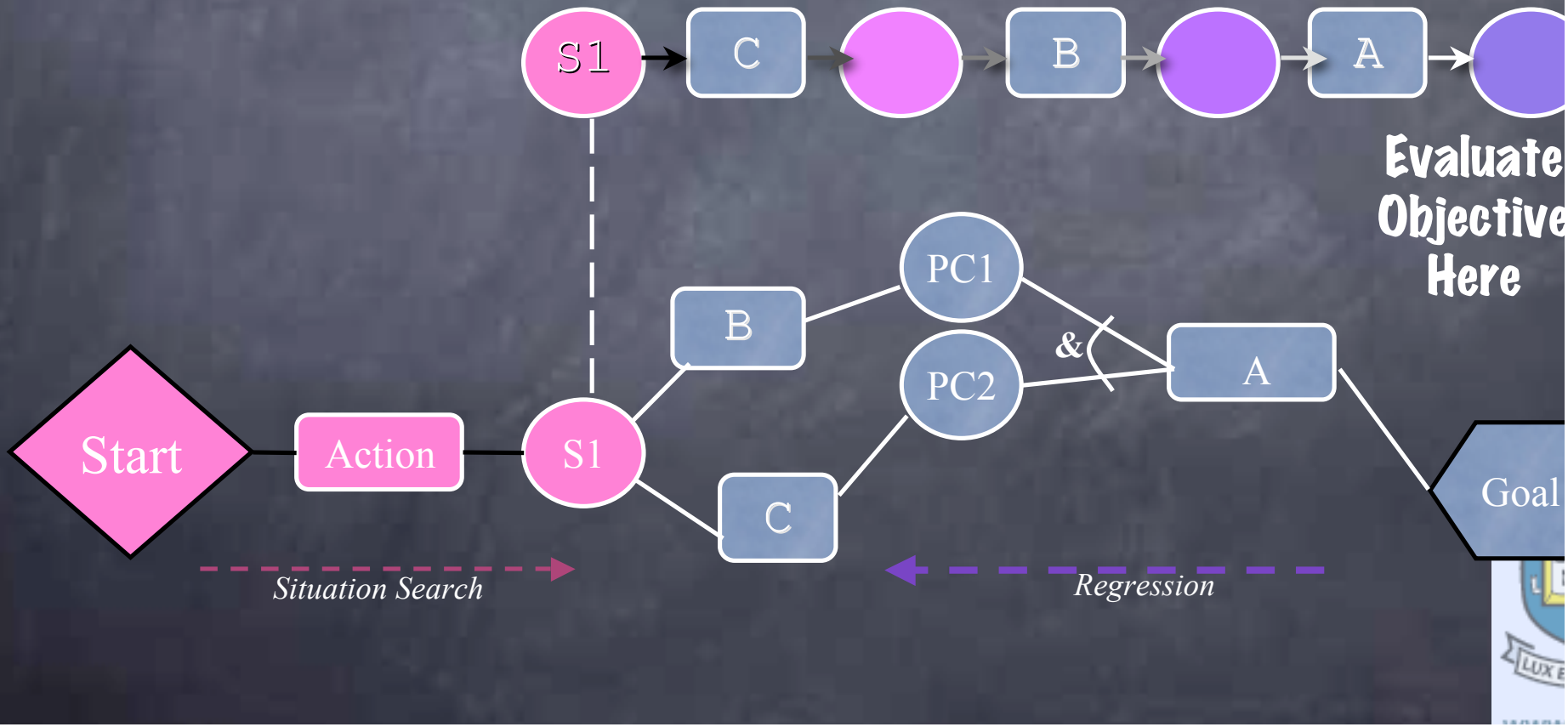
Goal

Estimated-Regression Planning

- At every situation, planner must compute a *regression-match graph* to estimate which actions are feasible and promising.
- Regression-Match graph is a modified AND/OR graph:
  - OR for the alternative actions for achieving a goal,
  - AND for action preconditions.

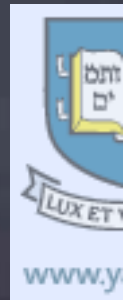


Evaluating Action C: Plausibly Project



Optop Algorithm

```
search-for-plan(prob, metric)
   let Q = queue of scored plan prefixes,
             initially containing only the empty prefix
     (while Q is not empty
       (Remove the plan prefix P with minimum score from Q;
        If P is a solution to the problem, return it
        Compute a regression-match graph relating the goal of pr
           to the situation reached after P, editing the regress
           match graph of P's predecessor if possible;
        For each action A recommended by the graph,
           Let R = some minimal reduction tree for A
             (Plausibly project R;
              Evaluate the metric in the resulting situation;
              Put the new prefix P+A on Q with that value as its
                score));
      return 'FAILED)
```



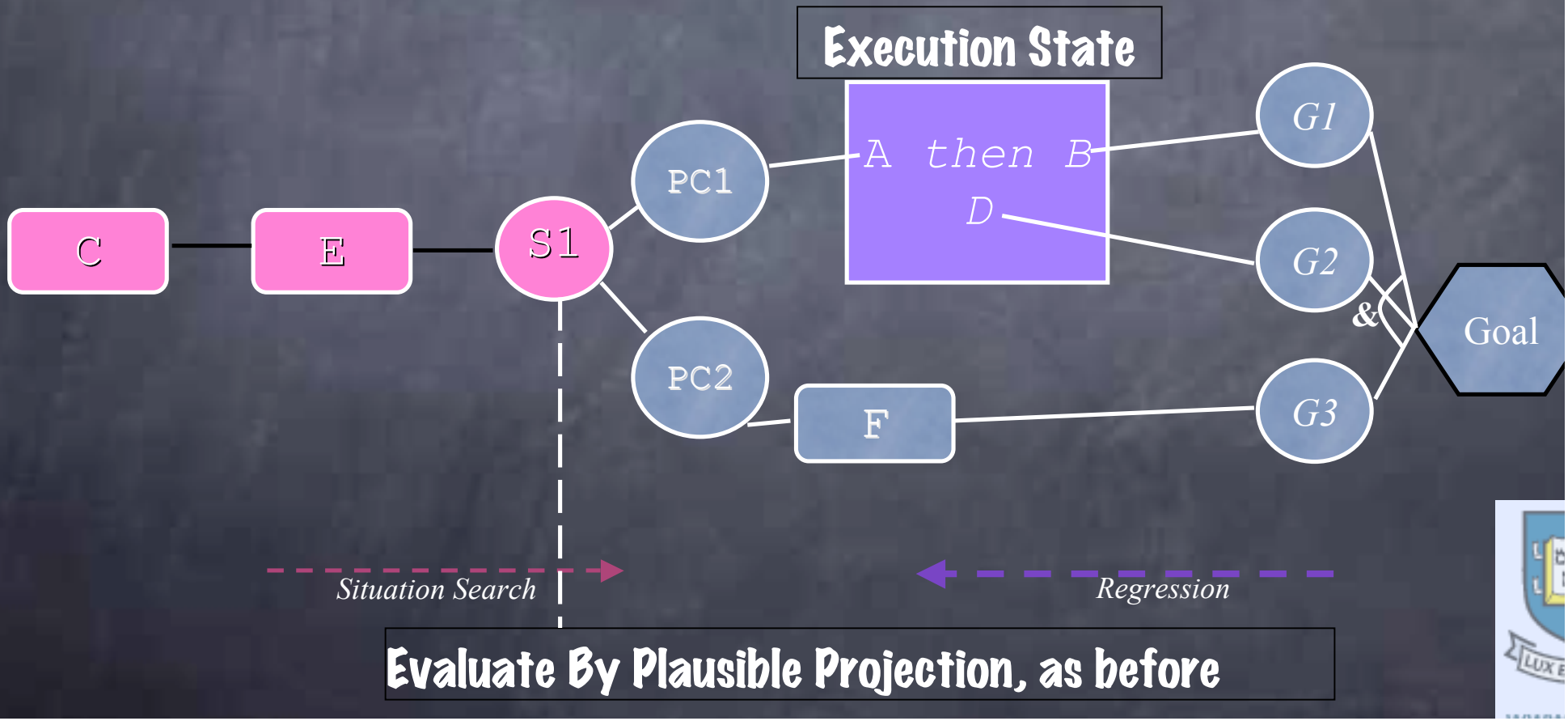
Changes to Handle Hierarchical plans

- View canned plan as a "resource"
- Composite actions are chosen for their overall effects
- New plan operator: select method for plan
- State representation includes partially executed HTNs.

Plan-State Representation

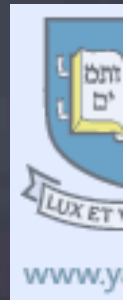
At some point we commit to a plan

```
(parallel (seq A B) (seq C D))
```



Conclusions: How to Enhance Expressivity

- The IPC and PDDL are working pretty well
- HTNs are an idea whose (notational) time has come.
- It isn't that hard to incorporate them into existing planning systems.



Conclusions: How to Enhance Expressivity

- Be maniacs! (Sometimes; usually idiocy is preferable.)
- Be elegant. Don't let current planner limitations overly constrain notation.
- Don't outrun the algorithms too much. If no one has a clue how to use the notation, it won't be used.
- Avoid committees and telecons :)