

# Grid-TLSE

A WEB EXPERTISE SITE FOR SPARSE LINEAR  
ALGEBRA

<http://www.enseeiht.fr/lima/tlse>

Members of the project

CERFACS, FERIA-IRIT, LaBRI, LIP, CNES, CEA, EADS, EDF, IFP

# Outline

## PART 1 (*Patrick Amestoy*)

- Introduction ( Motivation - Objective )
- Description of main components
- Running the Grid-TLSE demonstrator ( Illustration, Potential, Limitation)

## PART 2 (*Marc Pantel*)

- Description of the software components
- Concluding remarks
- List of open questions/problems  
— — > “Table ronde”

# Introduction

- 3-year project funded by ACI GRID program from the French Ministry of Research (January 2003 — — > January 2006)
- Research Labs : CERFACS and IRIT (Toulouse), LaBRI (Bordeaux), LIP-ENS (Lyon)
- Industrial partners :  
CNES, CEA, EADS, EDF, IFP
- International links : Berkeley, RAL, Parallab, Univ. Florida, Univ. Minneapolis, Univ. Minnesota, Univ. Tennessee, Univ. San Diego, Univ. Indiana, ...

# Goals

- Design a Web expert site for sparse matrices
- Limit our study to sparse direct solvers
- Disseminate our expertise
- Provide interface to experiment software
  - public . . . as well as commercial
  - sequential . . . as well as parallel
- Submit a matrix or use matrix collections
- Provide tools to help incorporating new services

# Example of expertise request

- Assumption : The performance (time and memory used) of our solvers depends mostly on the choice of the ordering used.
- Examples of request:
  - Memory required to factor a matrix
  - Error analysis as a function of the threshold pivoting value
  - Minimum time on a given computer to factor a given unsymmetric matrix ?

# GRID computing ?

- Each request involves a large number of elementary requests (e.g. as many simultaneous executions of a sparse package as available orderings or more generally appropriate values of input parameters)
- Choice of target computers depends on type of request: (Matrix availability, Memory requirement, CPU requirement, software availability, cost of computing time ...)
- Grid of moderate size where each elementary request will run on one node (mono or multiprocessor) of the Grid.

# Is it realistic ?

- Independency of elementary requests
- Results from experiments : synthetic data  
(*the expert site is not a computing engine*)
- Time to answer is not so critical
- Data persistency between elementary requests easier to express

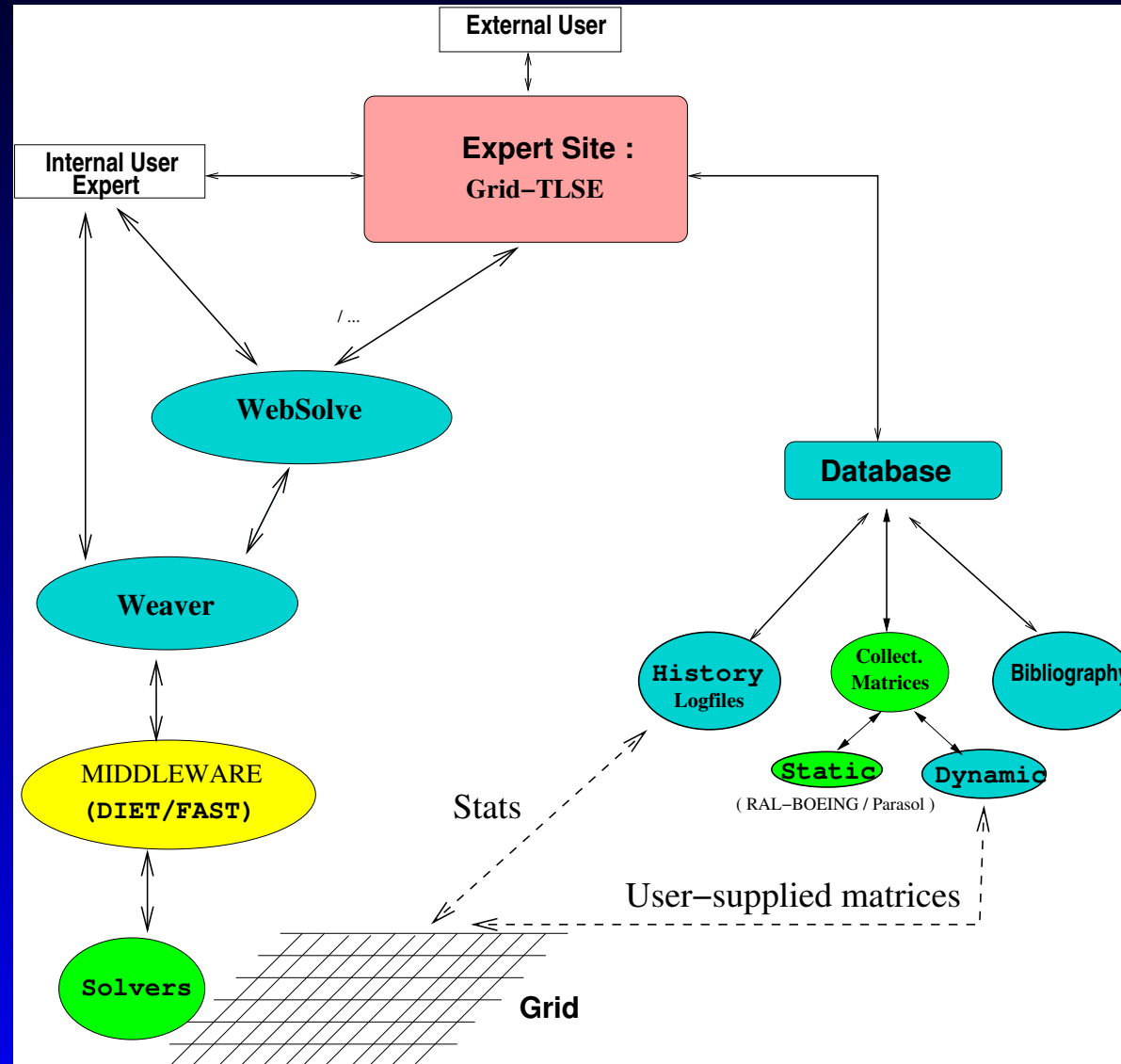
# Main components of the site

- Sparse matrix software: direct solvers
  - MUMPS (CERFACS, IRIT, LIP-ENS)
  - PaStiX, SCOTCH (LaBRI)
  - HSL (RAL) Library
  - SuperLU (Berkeley), UMFPACK (Univ Florida)
  - Others ...
- Database:
  - Bibliography
  - Sparse matrix collections (RAL-BOEING, PARASOL, user-provided)
  - Experimental results

# Grid Infrastructure

- Use of tools developed within GRID-ASP project (LIP-ReMAP, LORIA-Résédas, LIFC-SDRP) : **FAST, DIET**
- High-level administrator interface for the definition, the deployment, and the exploitation of services over a grid : **Weaver**
- Interactive Web interface with the Grid: **WebSolve**
- We do not provide computational resources, we just perform expertise (i.e. we may only report statistics on using various software on a matrix)

# Software components



# The Grid-TLSE demonstrator

- April 2003 : Presented at IPDPS (International Parallel and Distributed Symposium, Nice)
- Illustrate main functionalities
- Better understand difficulties,
- Get feedback from users and sparse community to
  - improve specifications and
  - validate software/middleware choices.

# Scenario : Ordering Sensitivity

- Algorithm
  - 1/ Get orderings
  - 2/ Obtain values of required metrics for each ordering
- Metrics of type **estimation**
  - 1 solver (get all internal orderings)
  - $\geq 1$  solver (get all possible orderings)
- Metrics of type **effective**

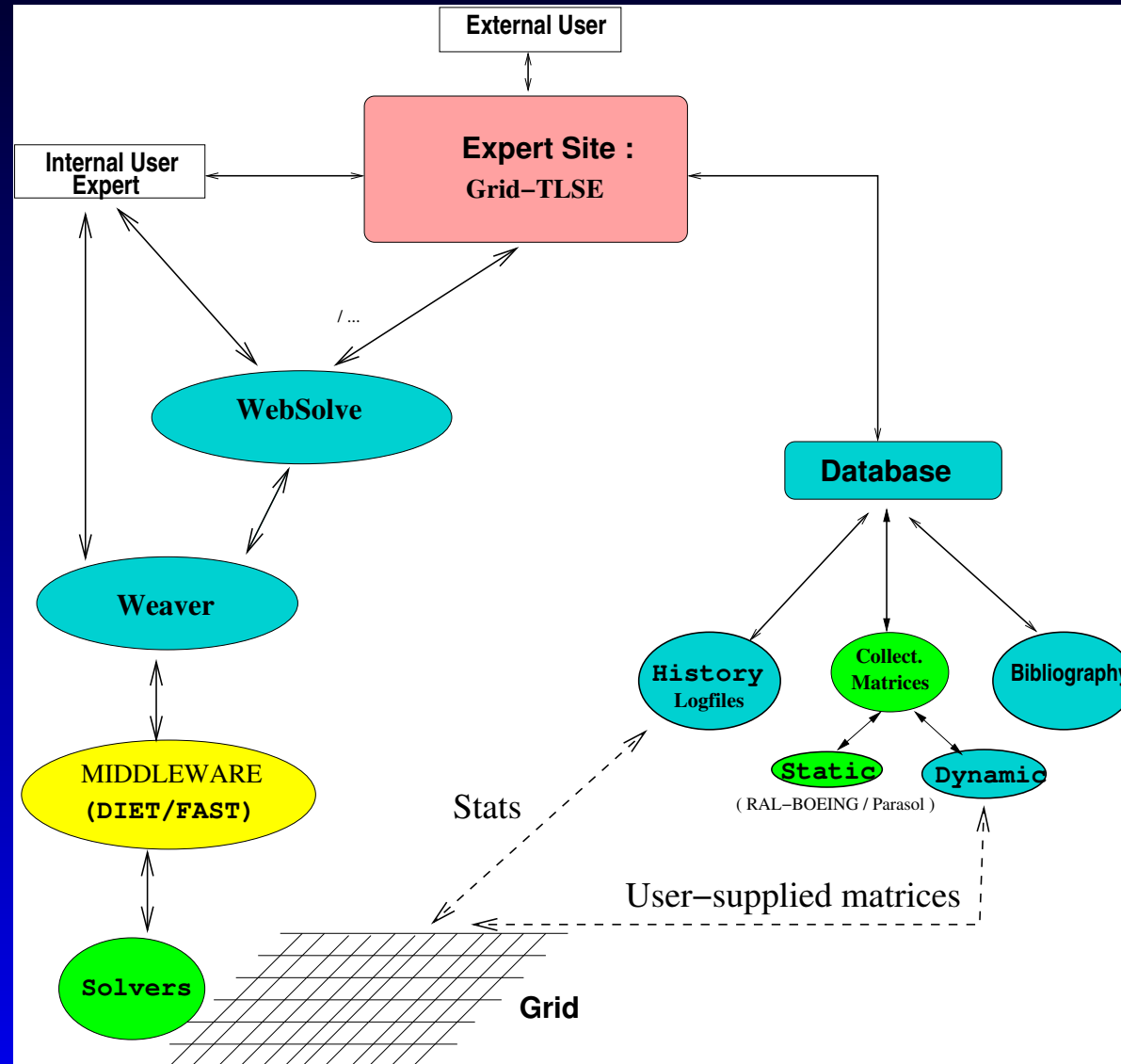
Factorisation also performed for each ordering

# Scenario : Minimum Time

## ALGORITHM

- **Phase 1:** Get **permutations** from all **solvers**.
- **Phase 2:**  
For each **permutation** and requested **solver**
  - Perform Flops estimation,
  - Keep best **permutation** per **solver**.
- **Phase 3:** For each **solver**:
  - Factorize with BOTH selected **permutation** and internal default **permutation**
  - Report statistics with minimum time.

# Software components



# WebSolve

Provide a simple and intuitive web interface

- For users :
  - Problem description ;
  - Request management ;
  - Graphical results
- And experts :
  - Users management ;
  - Requests management ;
  - Scenarii description and testing ;
  - Solvers description and testing ;
  - Grid state observation

# Weaver

Manage descriptions of :

- solvers ;
- architectures ;
- scenarii

Convert expertise requests to solver runs :

- Apply scenarii
- Look for solvers and architectures (trader service)

Abstracts from specific middleware (currently DIET)

# Weaver : Main problem

Solvers are quite complex :

- Many algorithmic control parameters
- Many secondary results qualifying an execution
- Several functions : Analysis, Factorisation, Solve, ...
- Highly dependant of the underlying architecture

All solvers share the same purpose

But their parameters and results can be quite different

# Solver profile

Property-based abstract description of all solvers

- Name ;
- Algorithm ( $LU$ ,  $QR$ ,  $LDL^t$ , ... ) ;
- Parameters ( $A$ ,  $b$ , ... ) and results ( $x$ , ... ) ;
- Controls (ordering, threshold pivoting, ... ) ;
- Metrics (flops, memory, precision, ... ) ;
- Architecture

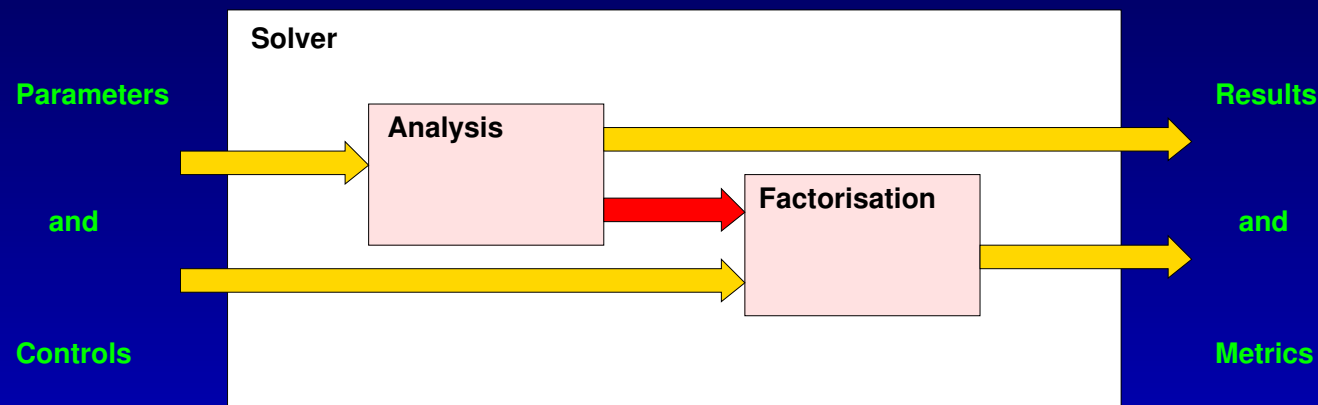
Each properties can be qualified *semantically*  
(symmetric matrix, 1 Gb of memory, ...)

**Critical** : Must be highly evolutive (meta-description)

# Solver signature

Description of a specific solver

- parameters and results ;
- hierarchical description of its sub-functions ;



- meaning of the profile properties ;

Some properties may not be significant

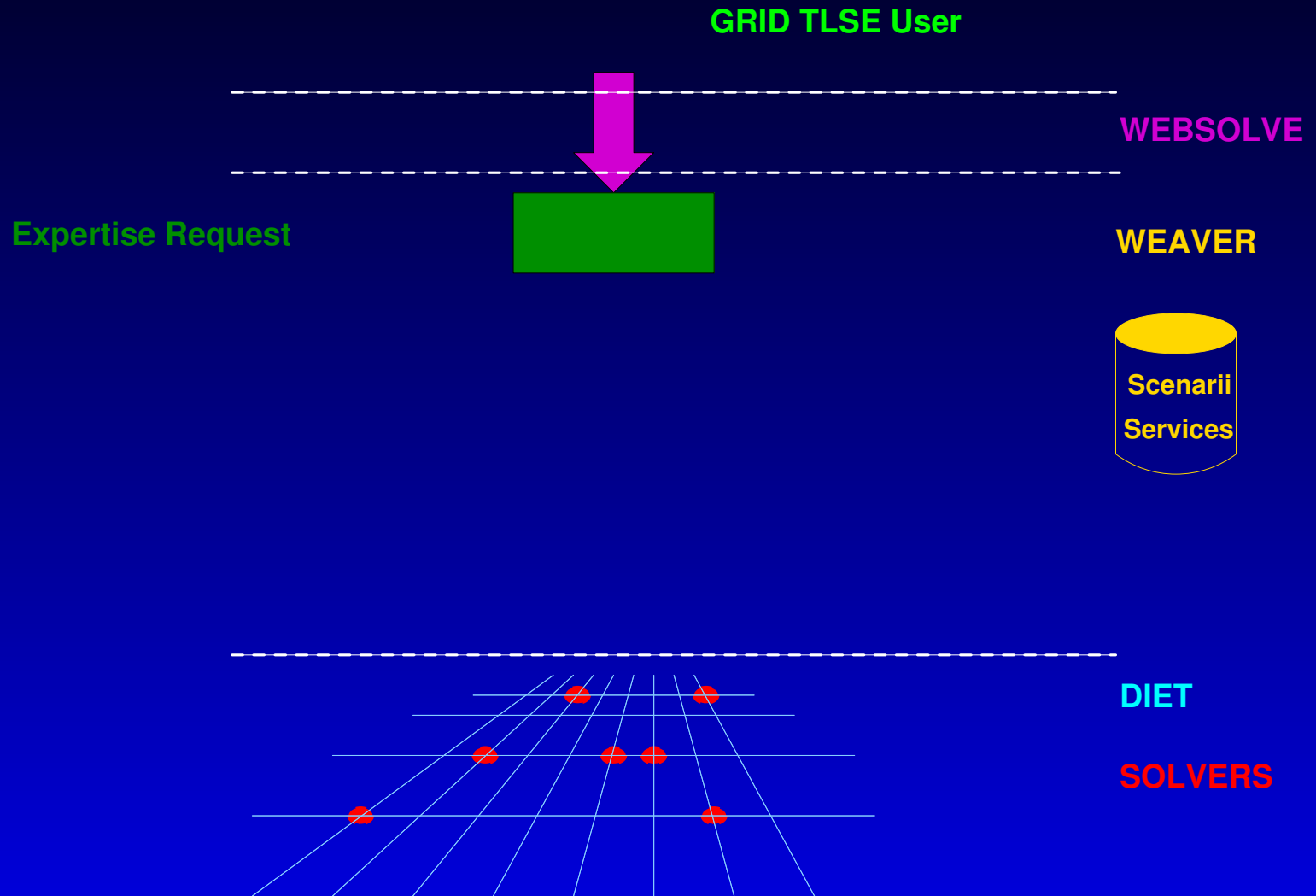
Some properties may be defined using other solvers

# Expertise requests

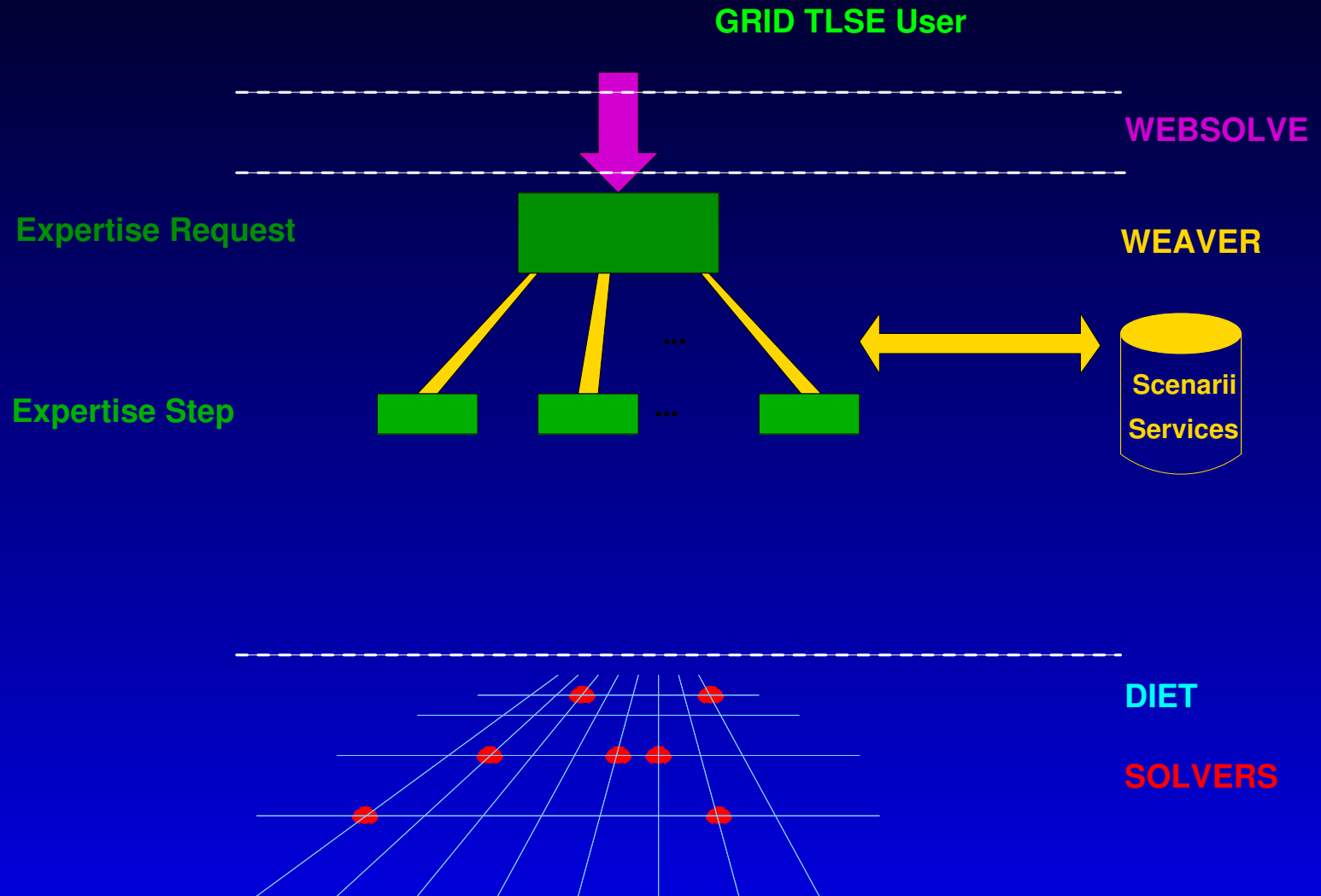
Problem defined by the user :

- Specify one (or more) expertise scenarii ;
- Provides values for some profile properties

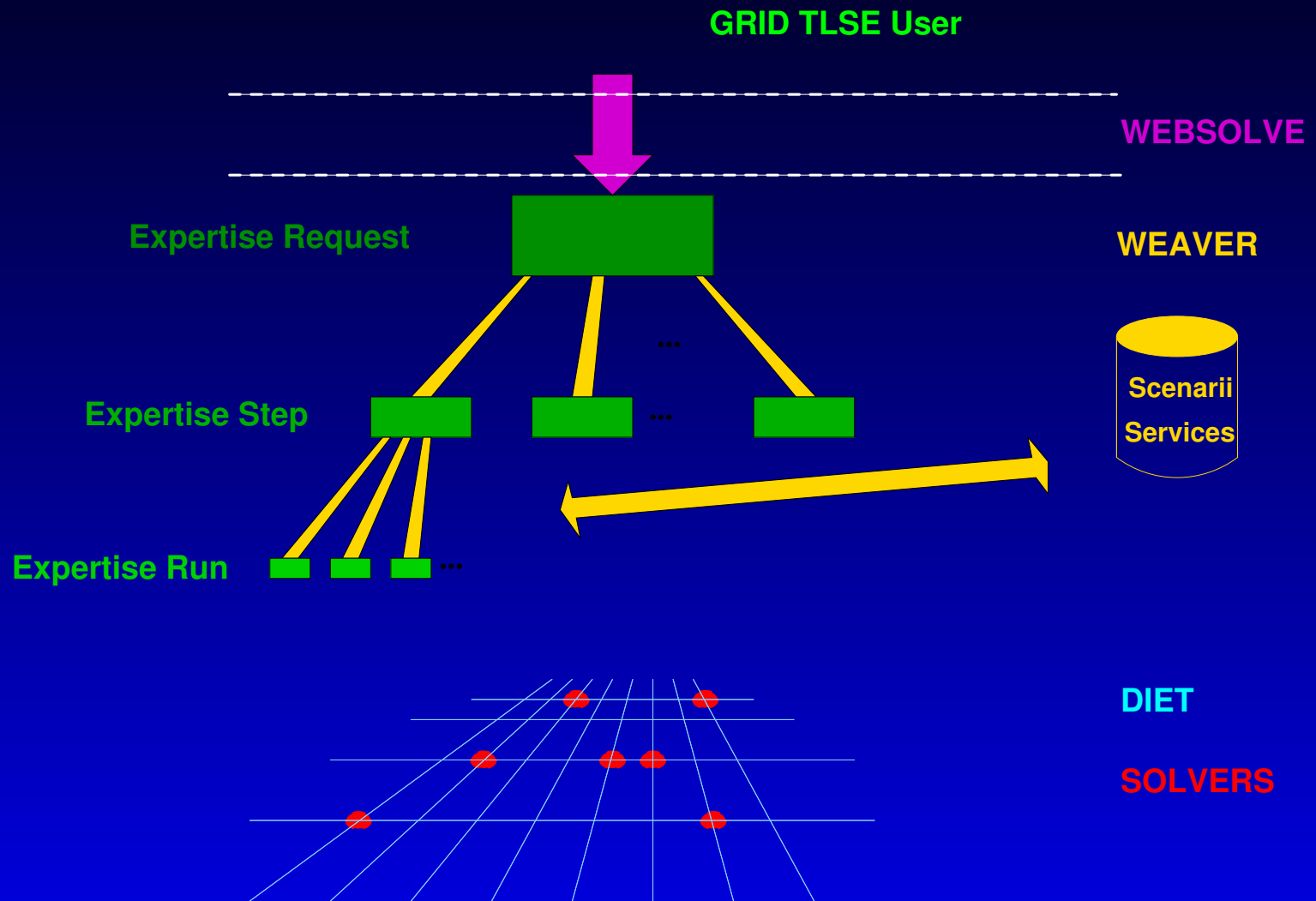
# Weaver prototype



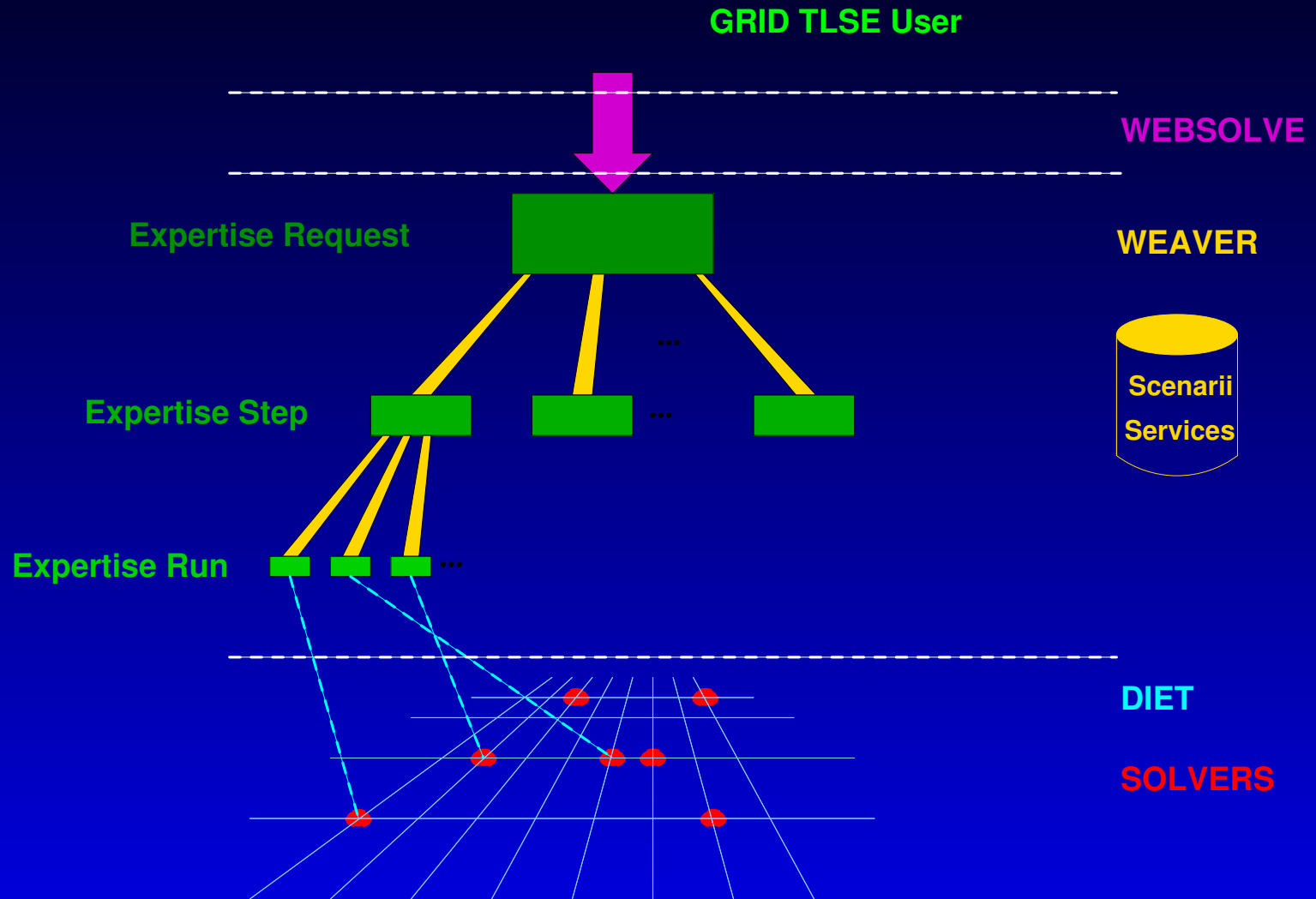
# Weaver prototype



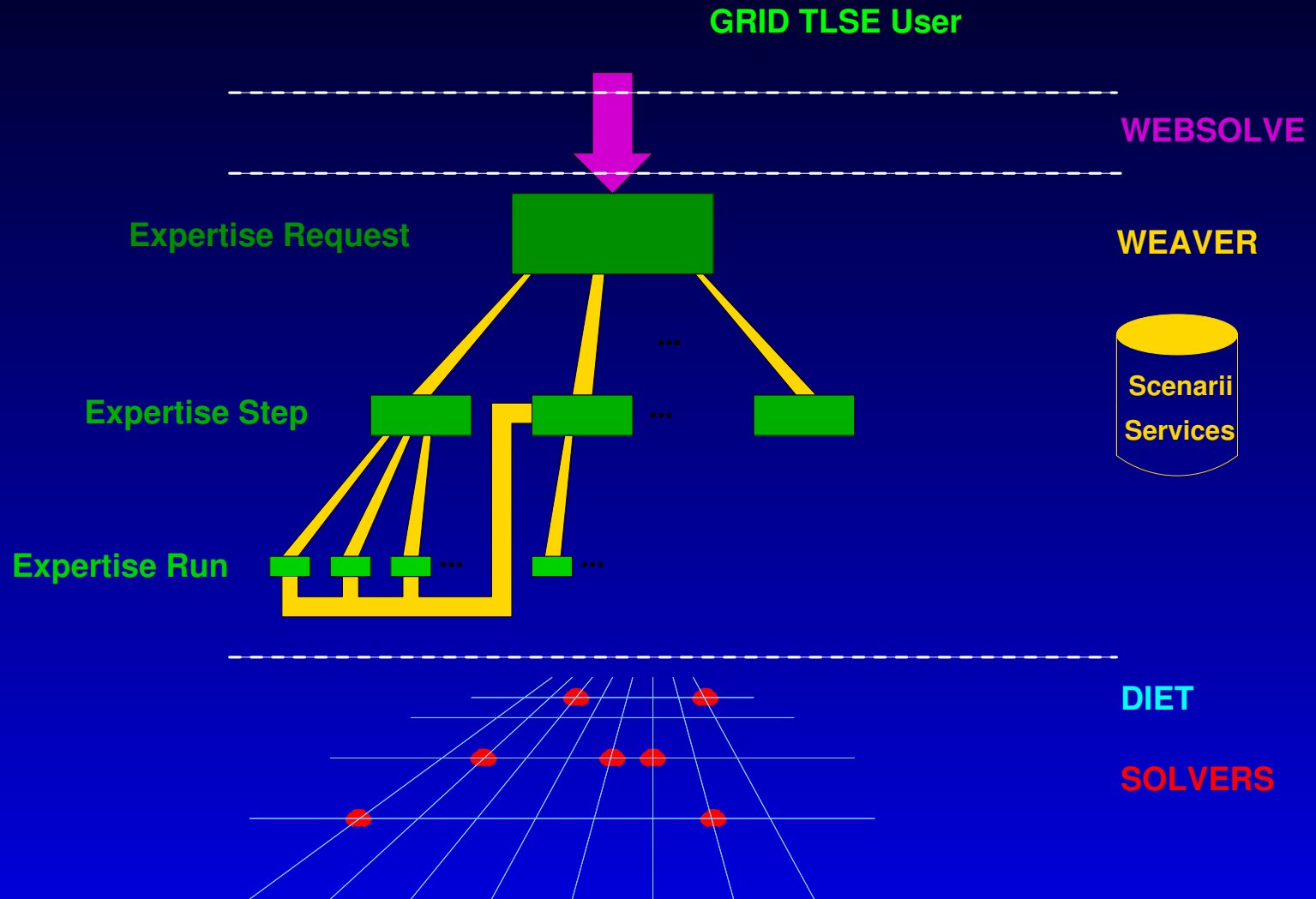
# Weaver prototype



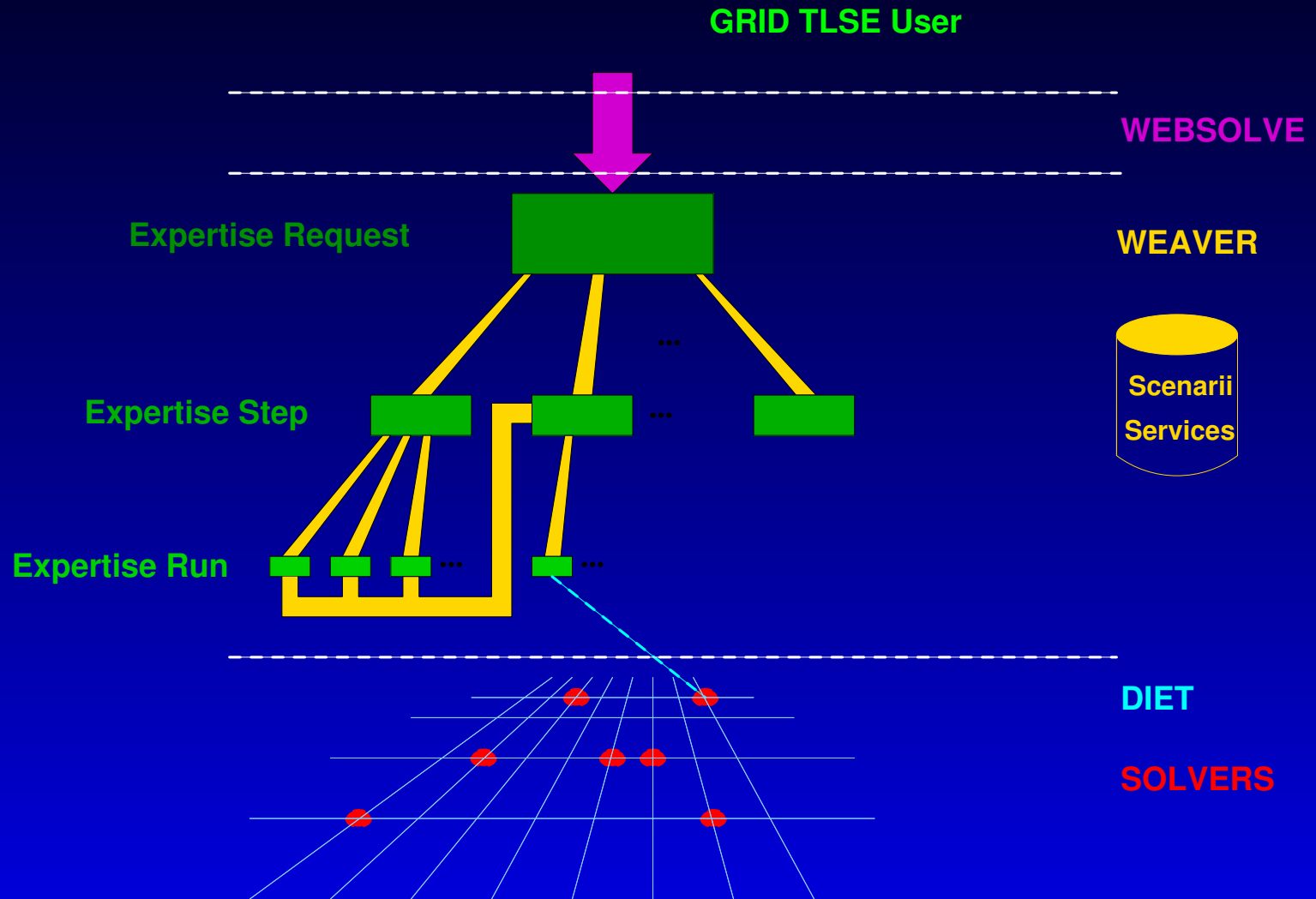
# Weaver prototype



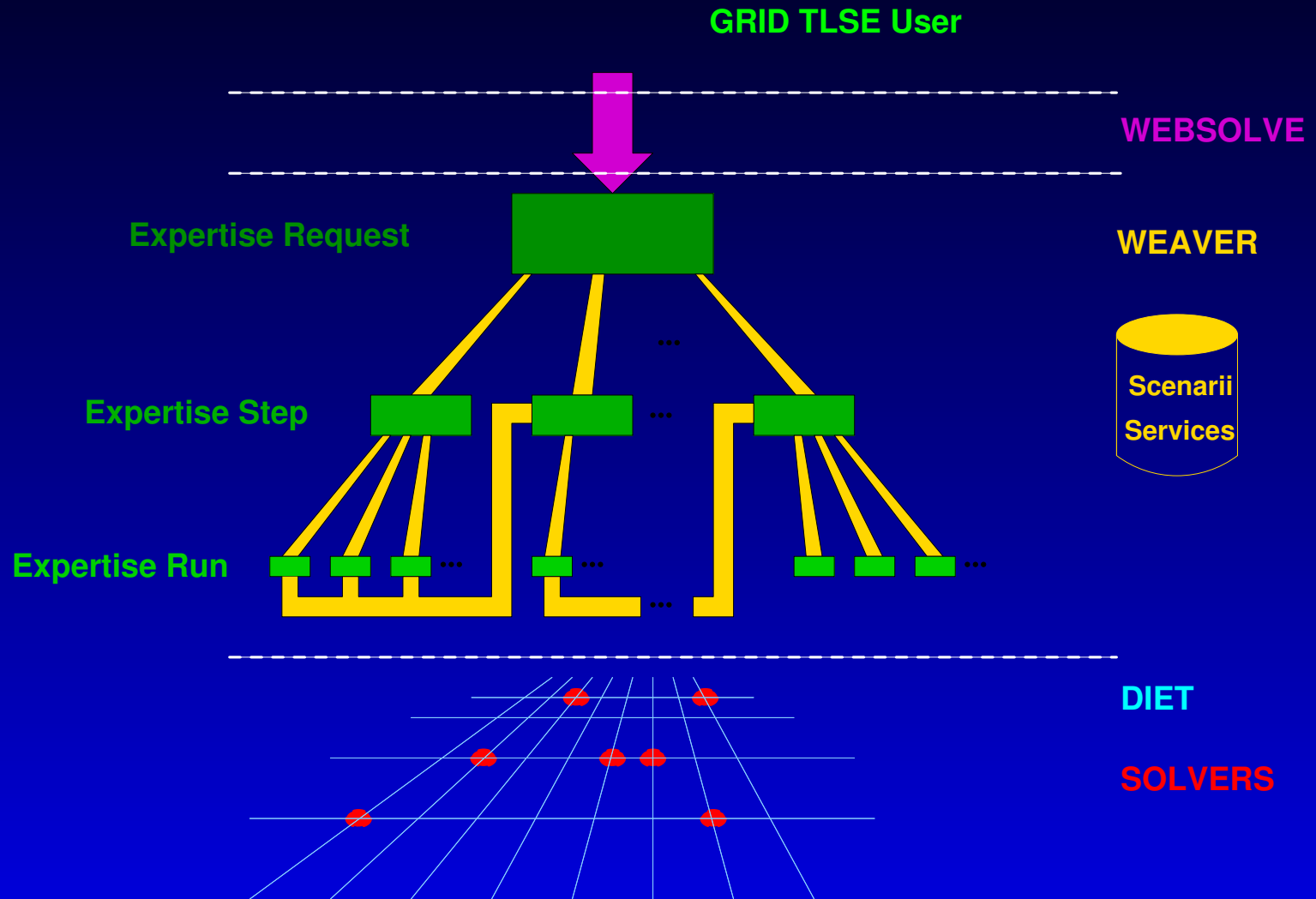
# Weaver prototype



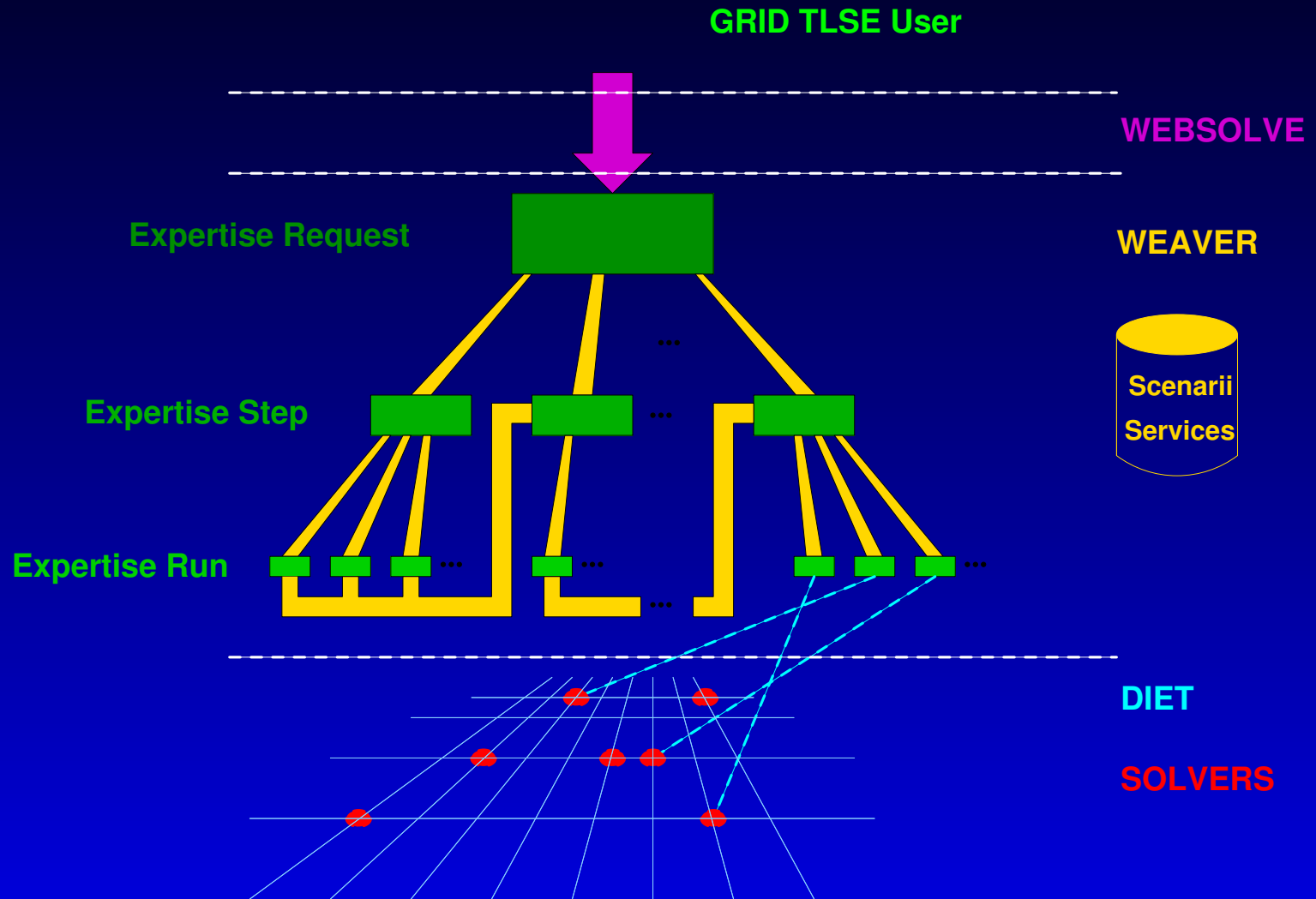
# Weaver prototype



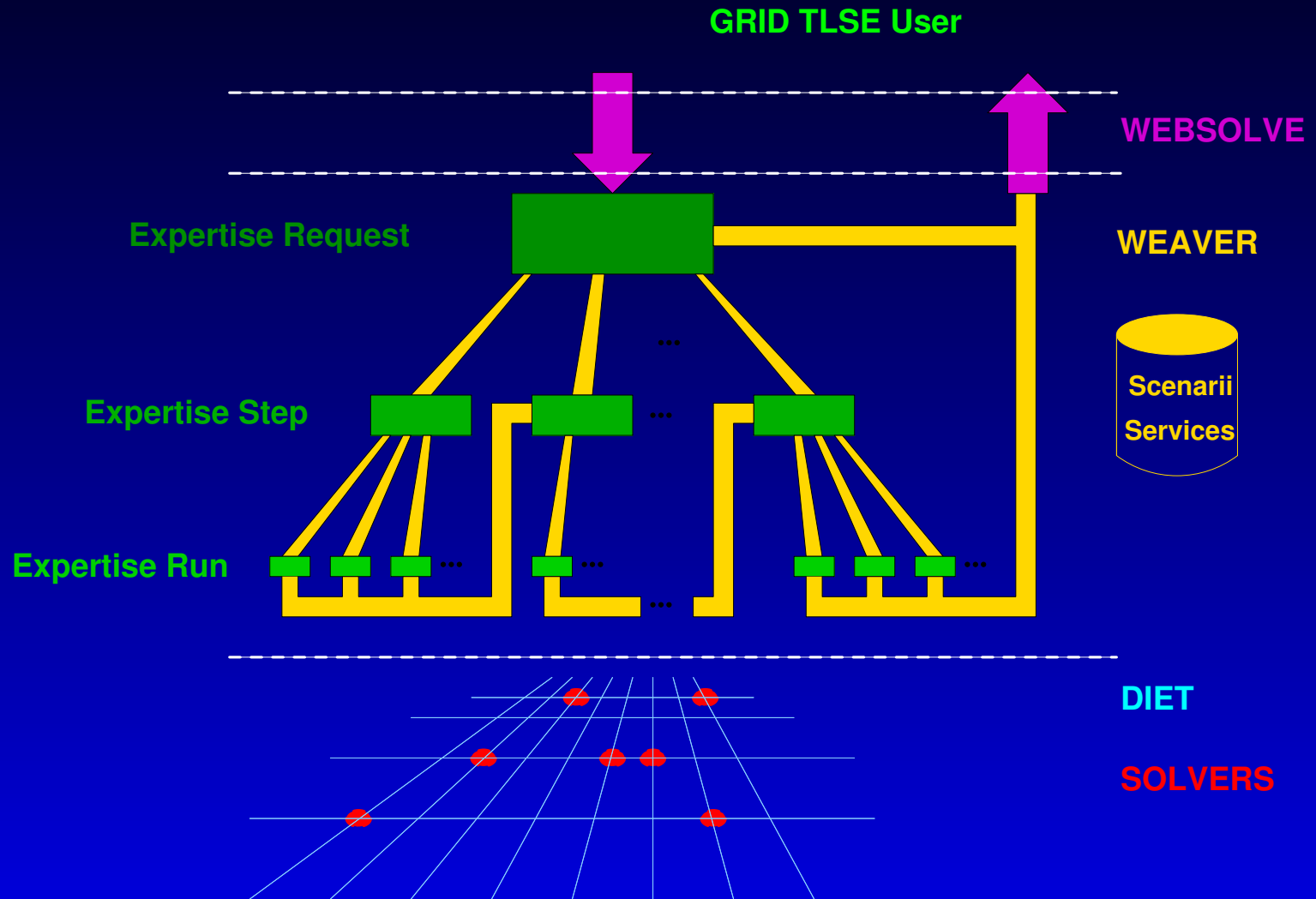
# Weaver prototype



# Weaver prototype



# Weaver prototype



# Current expert work

Writing JAVA classes to produce :

- Expertise steps from expertise requests (each scenario)
- Solver runs from expertise steps (each solver) :  
From abstract properties to concrete parameters and results
- Solver runs from previous step results (each solver) : idem
- Expertise graphs from expertise step results (each metrics)

Weaver purpose : reduce the amount of JAVA code written by the expert

# Future expert work

XML-based description (Web-based editor) of :

- Solver profile ;
- Solver signature ;
- Links between profile and signature ;
- Rule-based scenarii (expertise request pattern-matching)

**Important :** Profile evolution does not require any change in previous scenarii, although it may improve expertise quality

# Open issues

- Reasonable cost to add a solver, a scenario, some graphs ?
- Scheduling improvement based on signature hierarchical structure ?  
(persistence between runs)
- Scheduling improvement based on expertise results ?  
(flops and memory estimation)
- Batch scheduler interface
- Significant architecture description
- Expertise for *iterative* methods ?
- Data-mining expertise from crude results ?