PACE
The Warwick Performance System

G.R. Nudd, E. Papaefstathiou, D.J. Kerbyson,

High Performance Systems Group

University of Warwick

THE UNIVERSITY OF WARWICK

- Background
- Major European Collaboration Project
- Application Driven
- **P**ERFORMANCE **A**NALYSIS & **C**HARACTERISATION **E**NVIRONMENT

Aim

Develop techniques to enable users / application developers to make efficient use of distributed resources

- Application Development
- System Resources
- On the fly *Optimization*

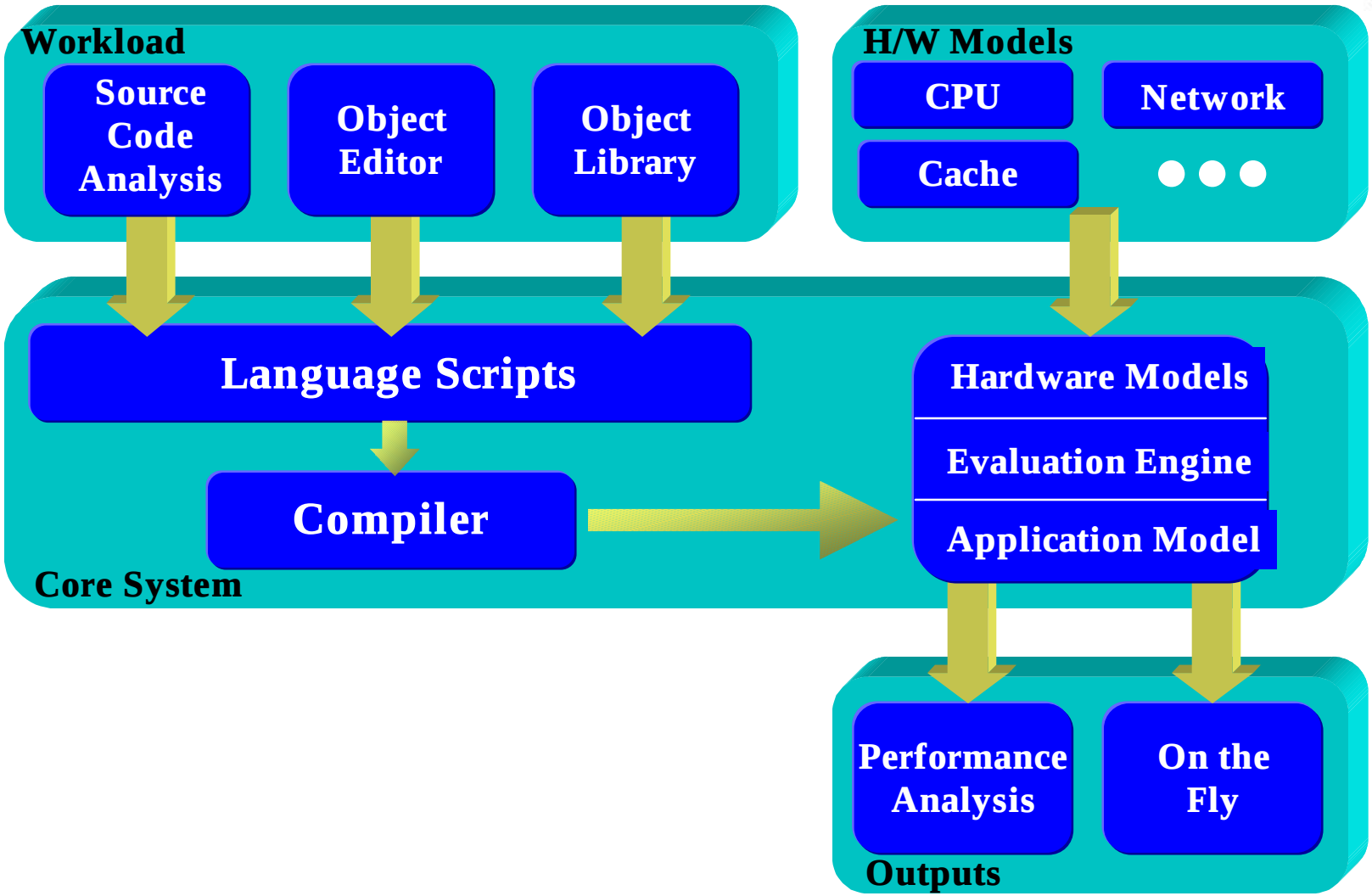
Applications

THE UNIVERSITY OF WARWICK

- Graphics (image synthesis)
- Image Processing
- Real Time Signal Processing
- Financial Codes

PACE

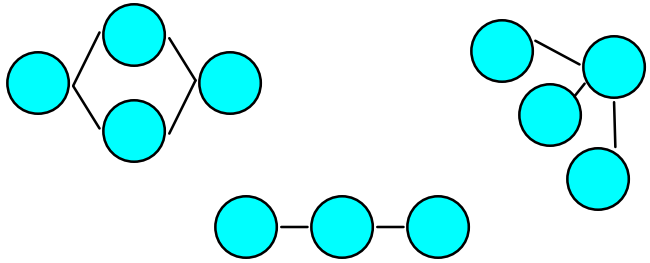
THE UNIVERSITY OF WARWICK



On-Going Research

- System Overview
  - Methodology
  - Performance Language
  - Application Workloads
- Models & Outputs
  - Model Development
  - System Outputs
  - On-the-fly & Scheduling

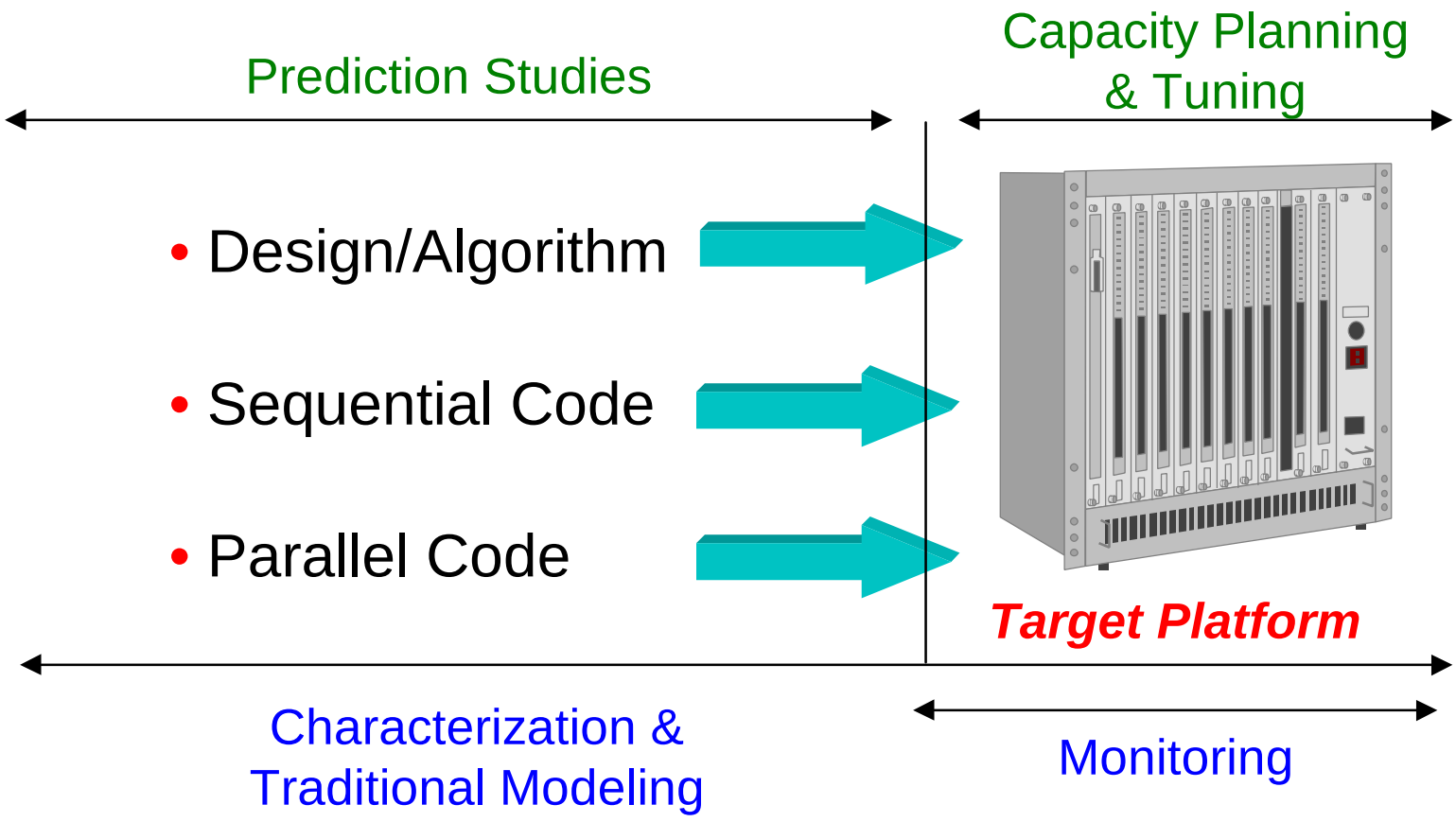
Performance Oriented Software Development (not!)



Parallelisation Strategies

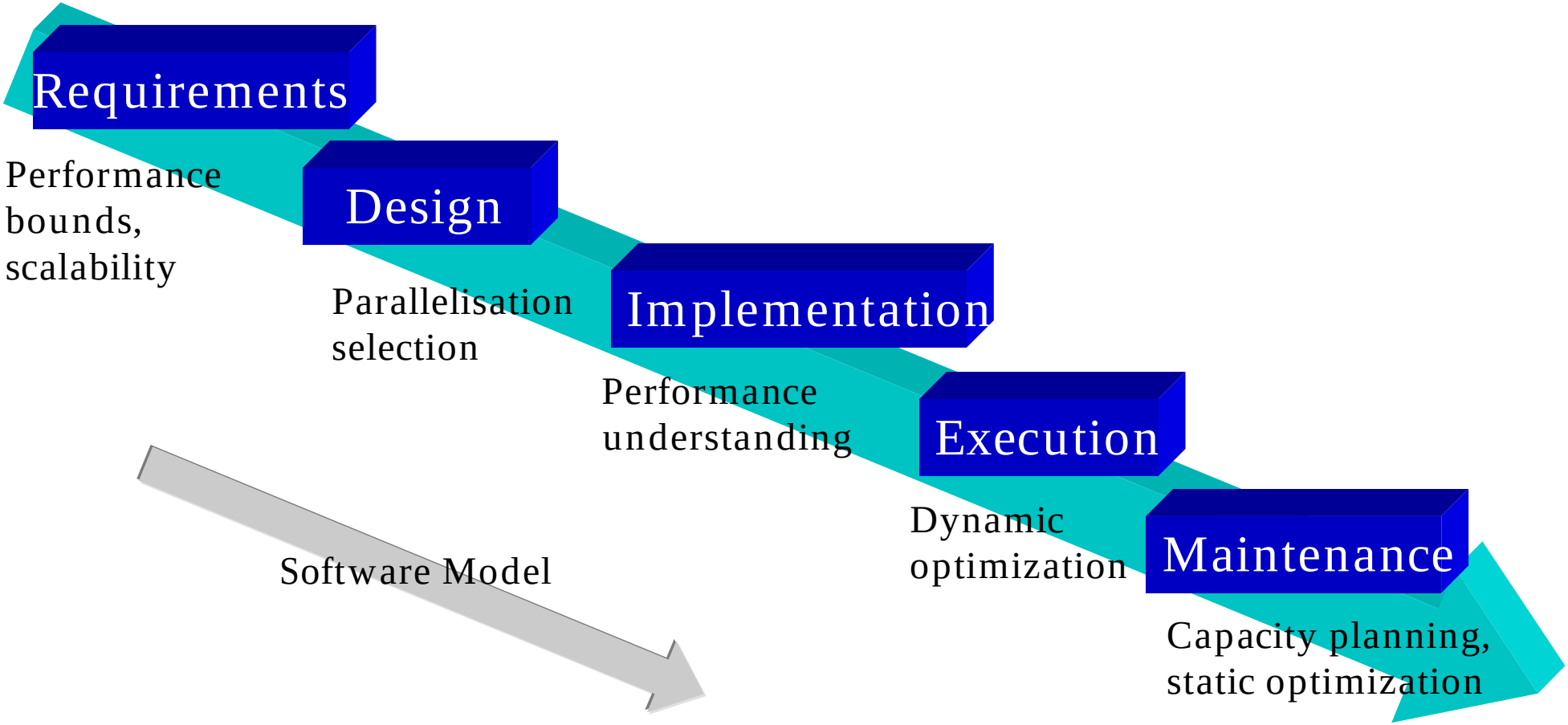
High Performance Systems

Software Performance Studies

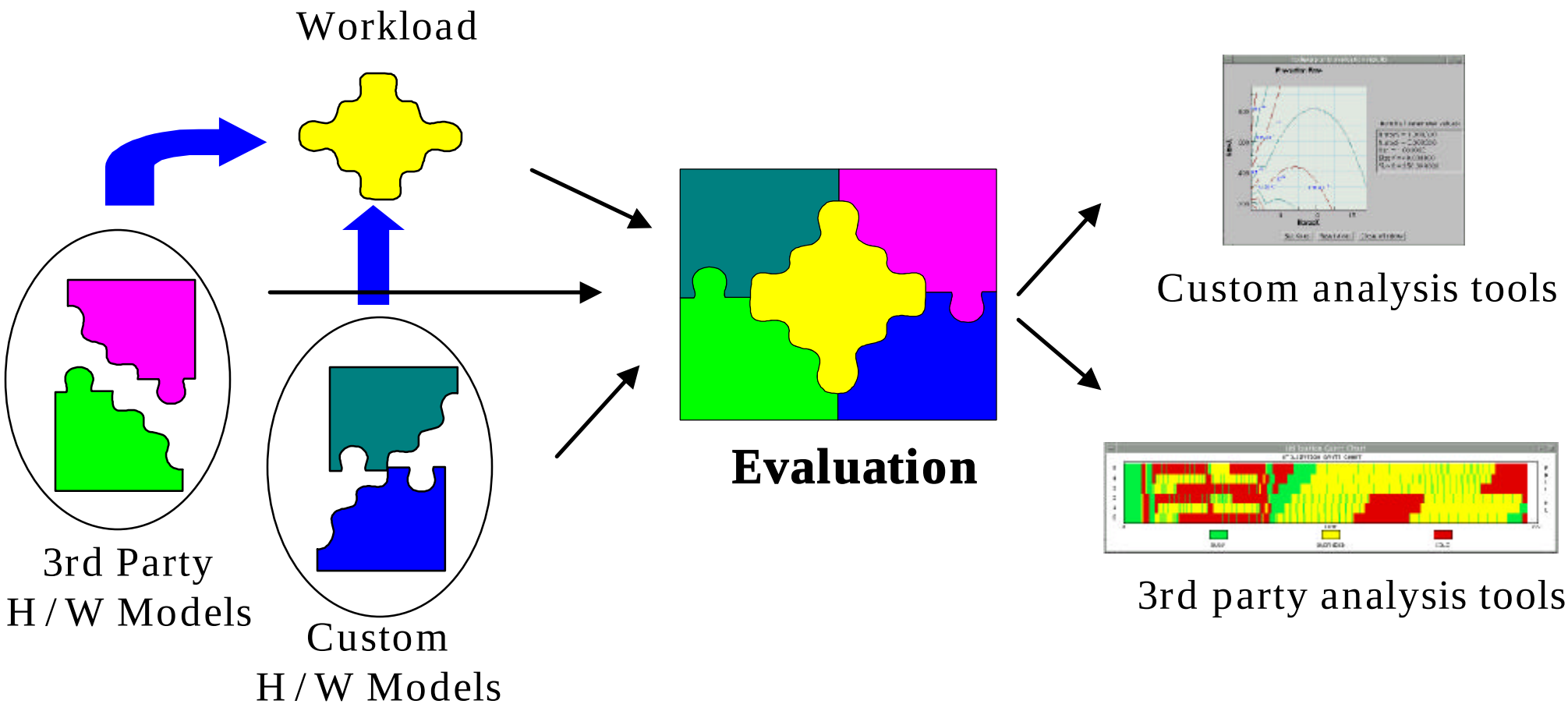


Goals

Support Software Lifecycle



Goals: Open System

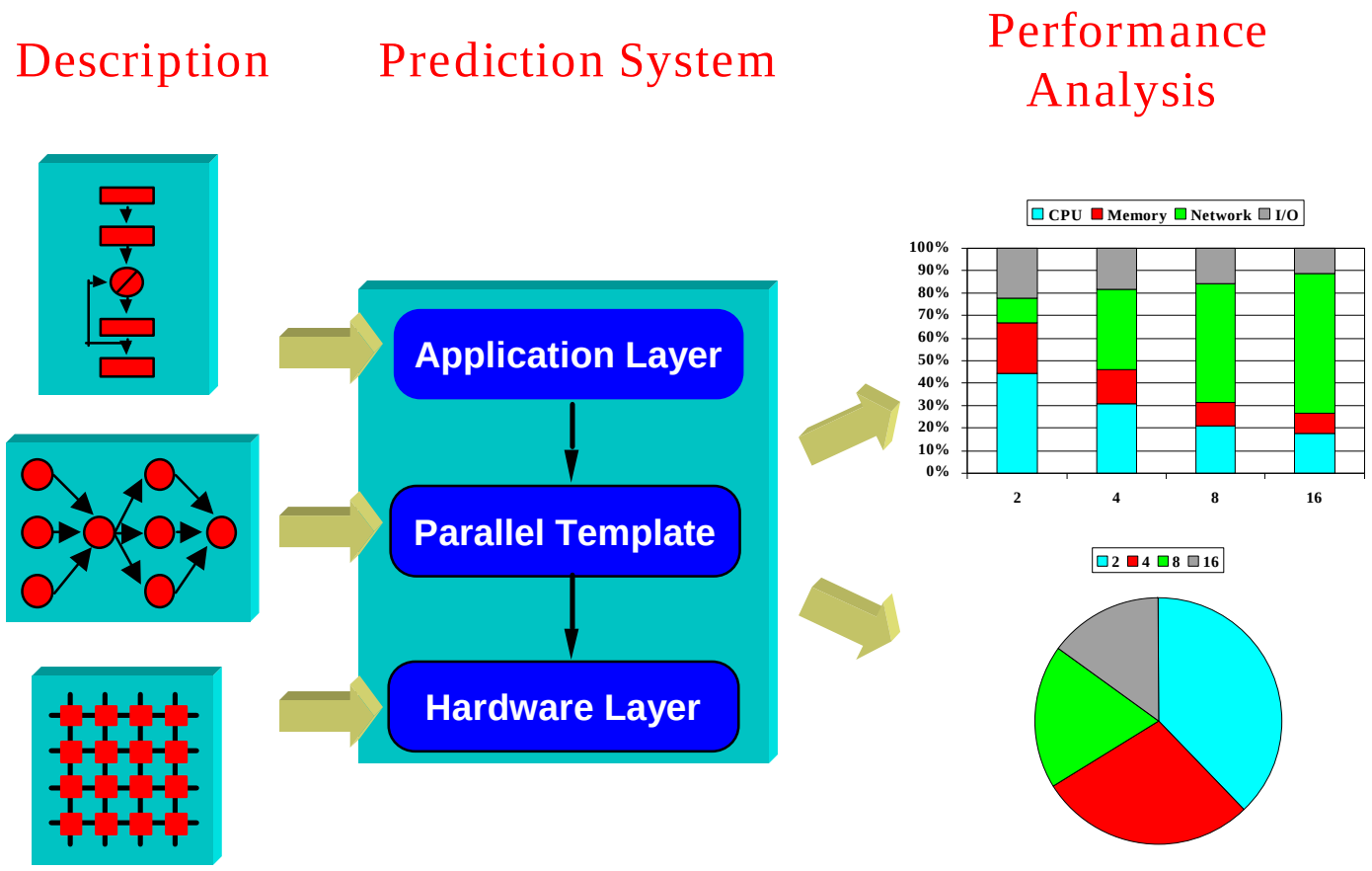


Goals

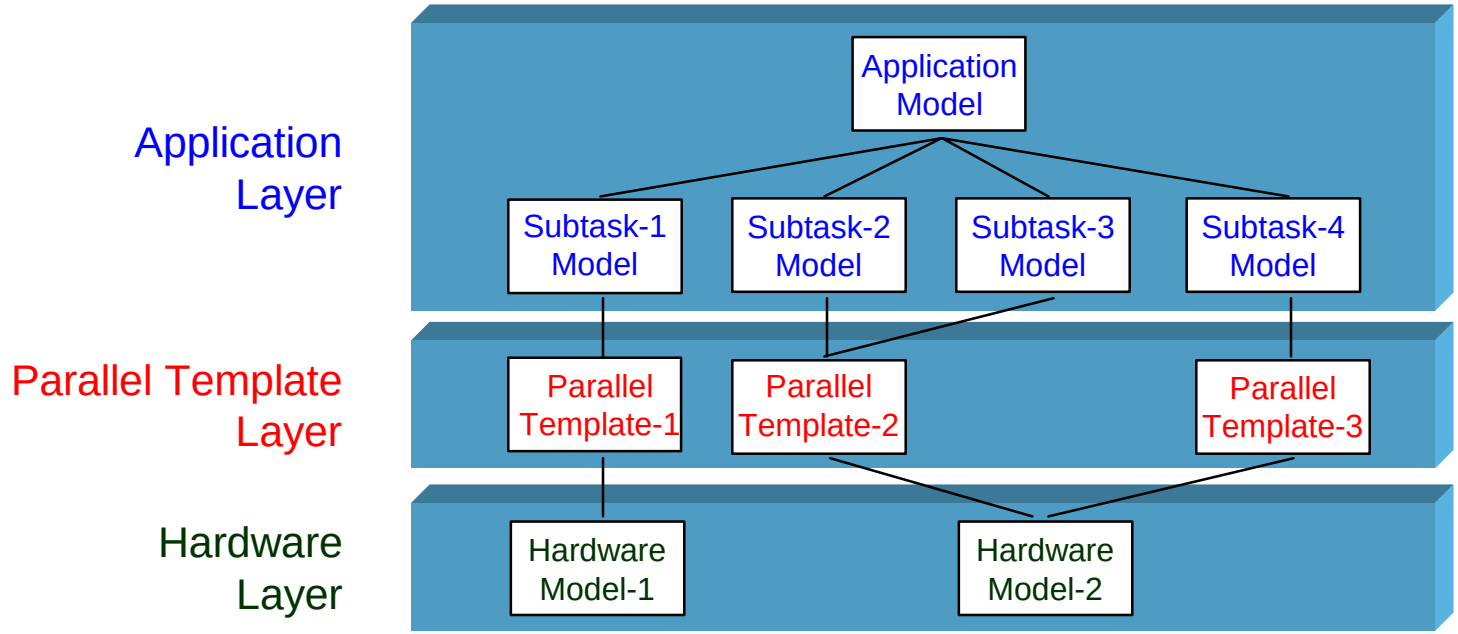
THE UNIVERSITY OF WARWICK

- Work with any system
- Multiple level of abstraction
  - Accuracy
  - Speed of evaluation
  - Model development effort
- Automation

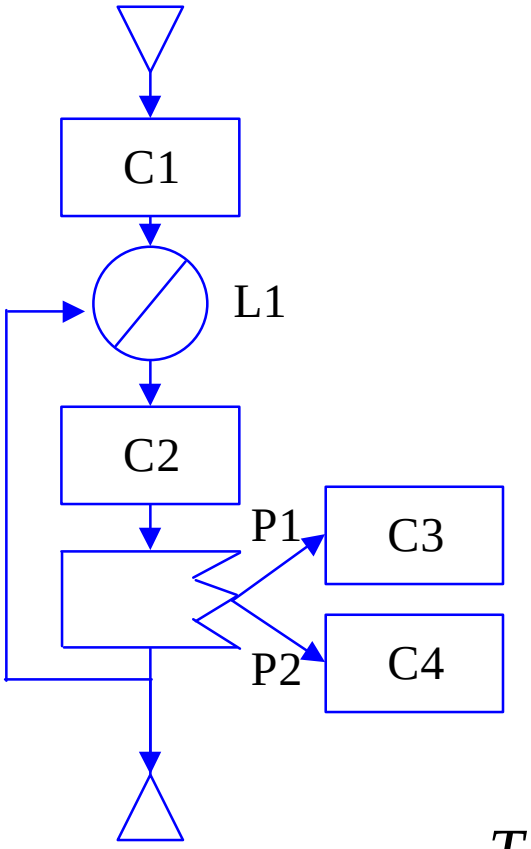
Layered Framework



Hierachical Layered Framework Diagrams (HLFD)



Application Layer



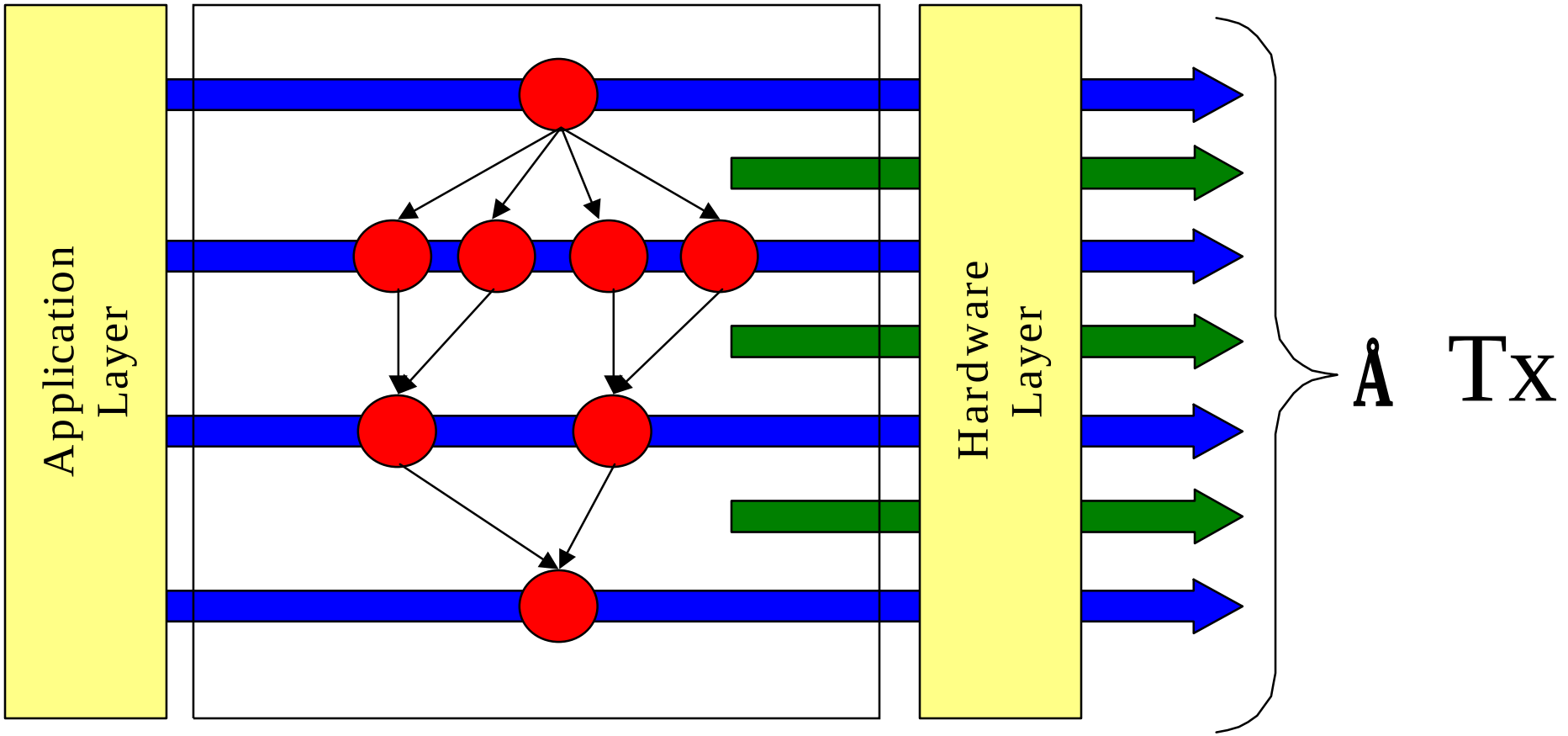
$$\rightarrow \; v_{C1} \; \rightarrow \; hm(v_{C1}) \; \rightarrow \; t_x$$

Resource Usage Vector
flops, language operations, instructions, memory references, etc

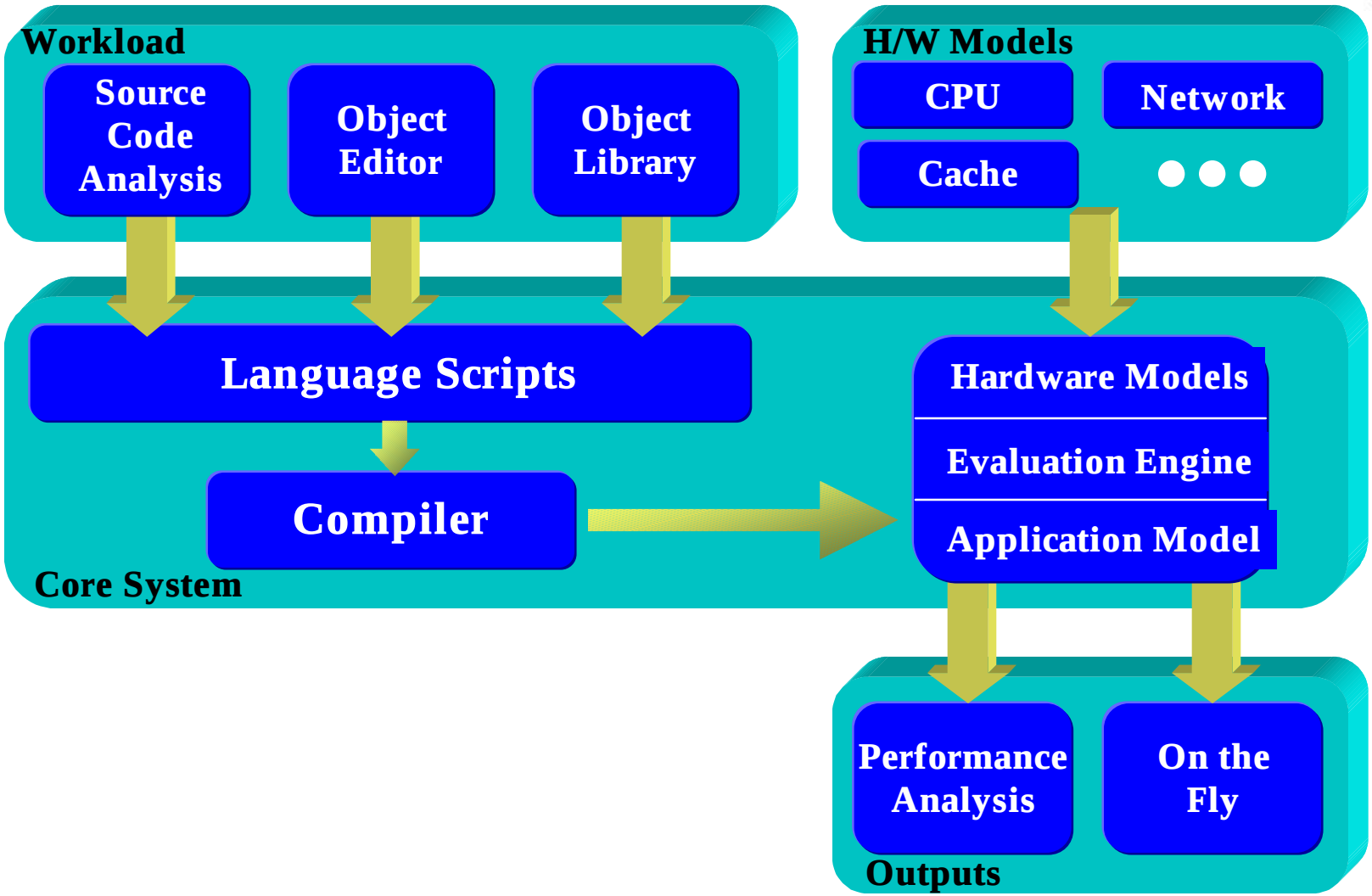
Hardware Model
statistical, analytical, simulation

$$T_x = hm(v_{C1}) + L1 \cdot (hm(v_{C2}) + P1 \cdot hm(v_{C3}) + P2 \cdot hm(v_{C4}))$$

Parallel Template



Overview of PACE

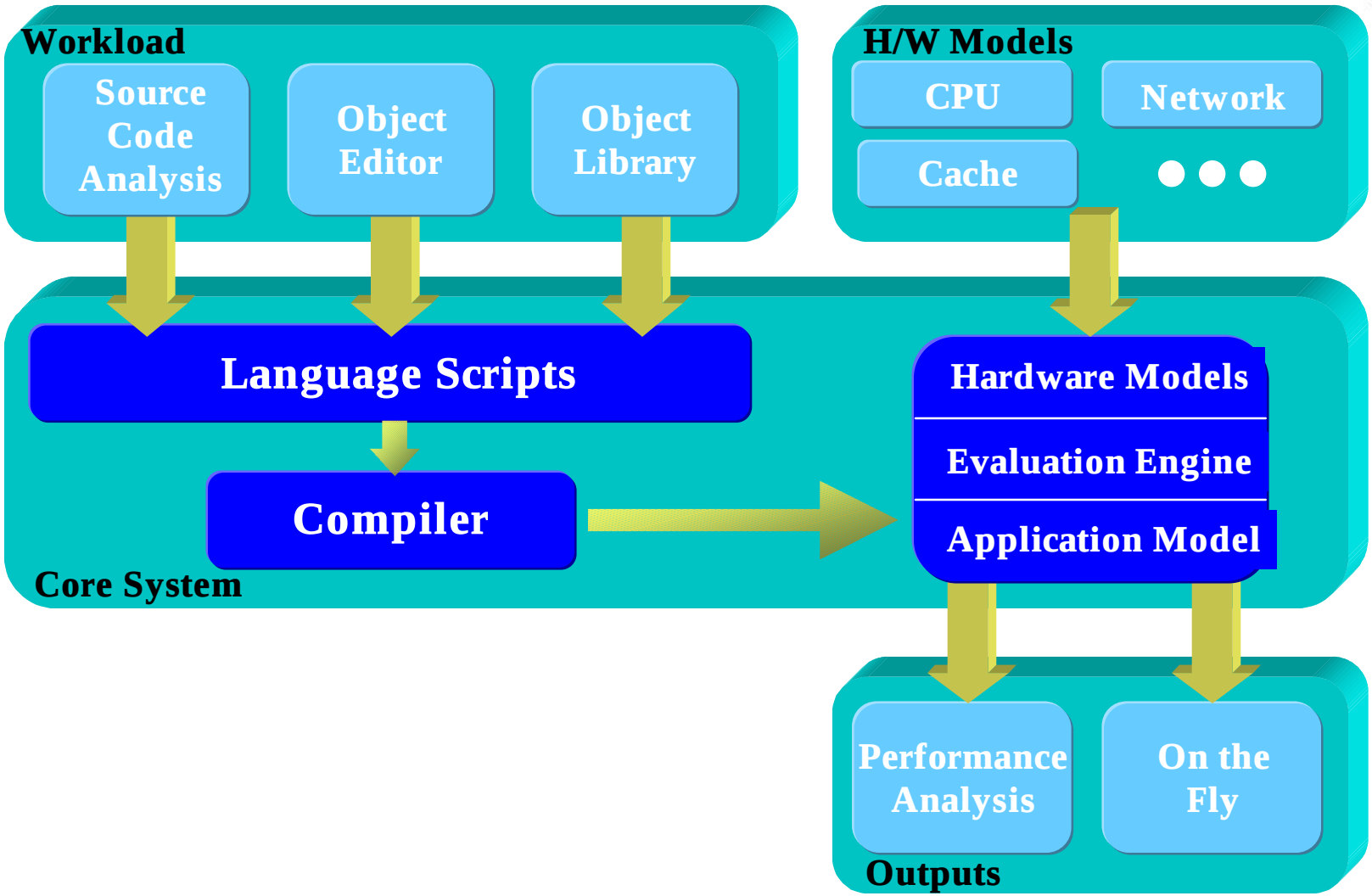


PACE

THE UNIVERSITY OF WARWICK

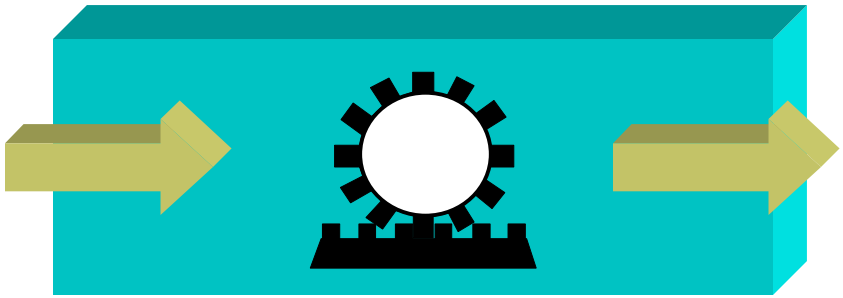
- Is based on a performance language
  - Control flow, parallelisation, resource usage
- Automated model creation
- Automated model evaluation
- Performance analysis environment
- Other support tools (e.g. source code analysis, object browser, visualization)

Core System


Workload
Source Code Analysis
Object Editor
Object Library
H/W Models
CPU
Network
Cache
Language Scripts
Compiler
Hardware Models
Evaluation Engine
Application Model
Core System
Performance Analysis
On the Fly
Outputs

Model Objects

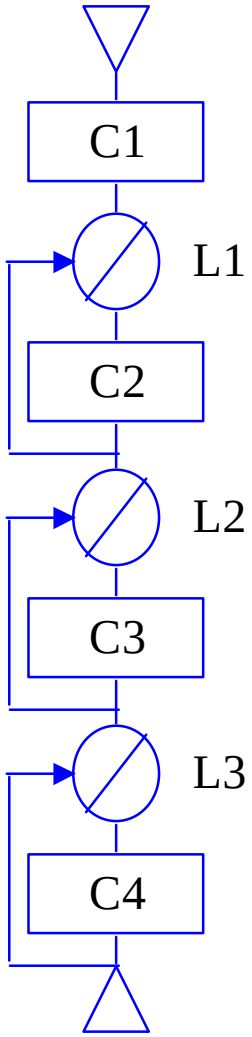
Model Object Structure



application	user input	system & problem setup	subtasks
subtask	problem size	computation (cflow)	partmp
partmp	problem size & computations	model config	hardware models

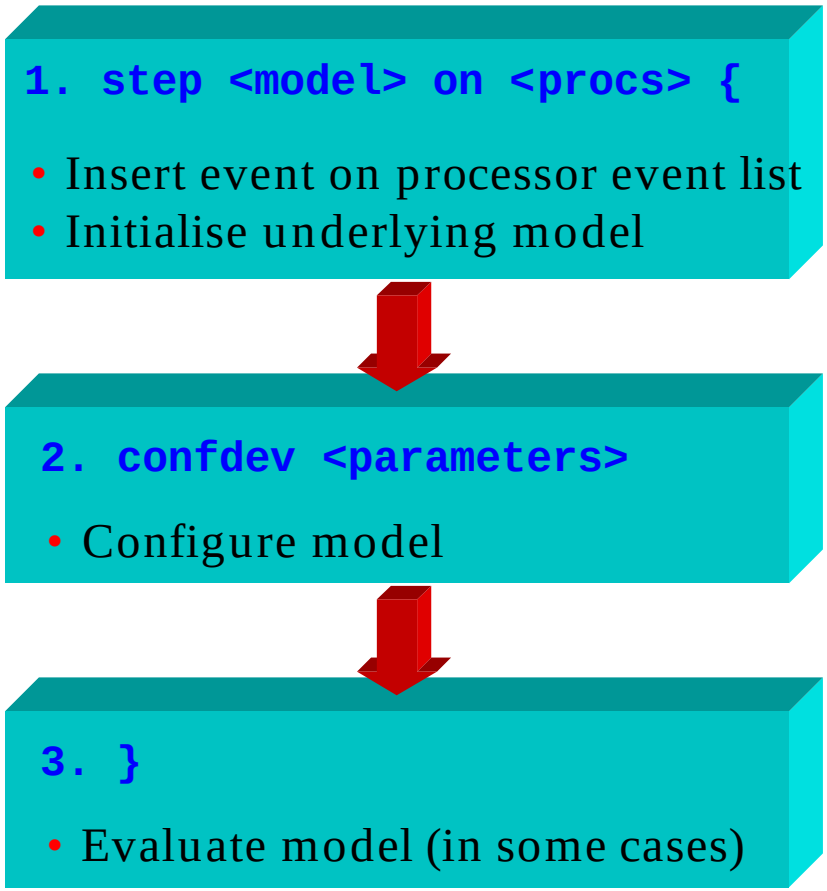
```
application improc {
    …
    link {
    histo:
       SizeX = ImgSzX/hardware.Nproc,
       SizeY = ImgSzY;
    …
    }

    proc exec init {
          …
          call histo;
          …
    }
    …
}
```

Control Flow Procedures



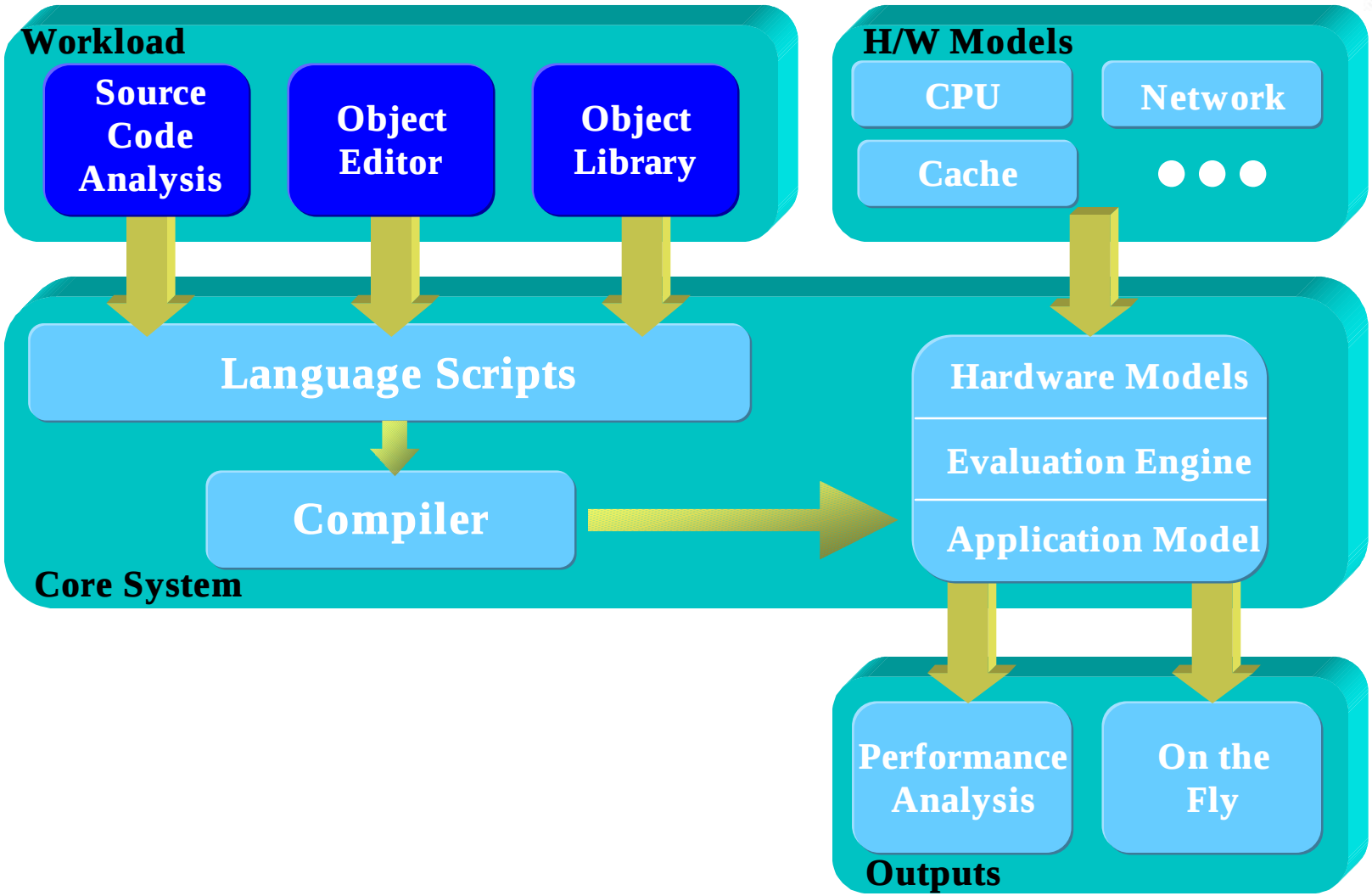

```
proc cflow Merge{
   var vector:
      start = <is ct, 1.47>,
      merge = <is suif, ldc_i32,add_i32>,
      tlcp = <is ct, 6.03>,
      otcp = <is clc, IASG, ICMP,
               2*IADD,4*VIDX, 2*FASG>;
   compute start;  (* C1 *)
   loop (<is clc, FOR>,2*dtpp*Pmrg) (* L1 *)
      compute merge; (* C2 *)
   loop (<is clc, FOR>,2*dtpp*(1-Pmrg)) (* L2 *)
       compute tlcp; (* C3 *)
   loop (<is clc, FOR>, dtpp) (* L3 *)
      compute otcp; (* C4 *)
}
```

Step and Confdev

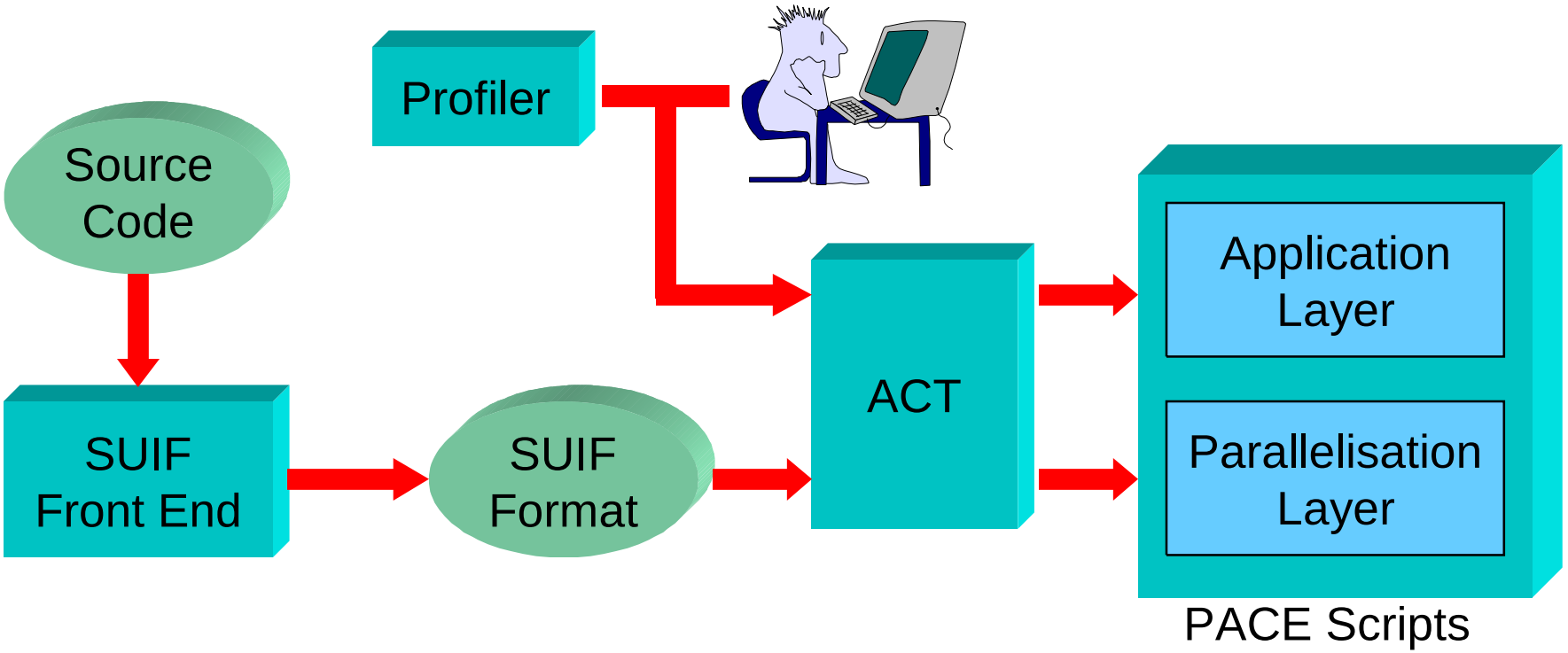



```
step cpu on 2,hardware.Nproc {
   confdev TxProc;
}

step pvminitsend on 2,hardware.Nproc {
   confdev PVM_DataDefault;
}

step pvmpack on 2,hardware.Nproc {
   confdev MsgSize,PVM_Int;
}

for( i = 2; i <= hardware.Nproc; i++ )
   step pvmcom {
      confdev i,1;
   }
```

Workload

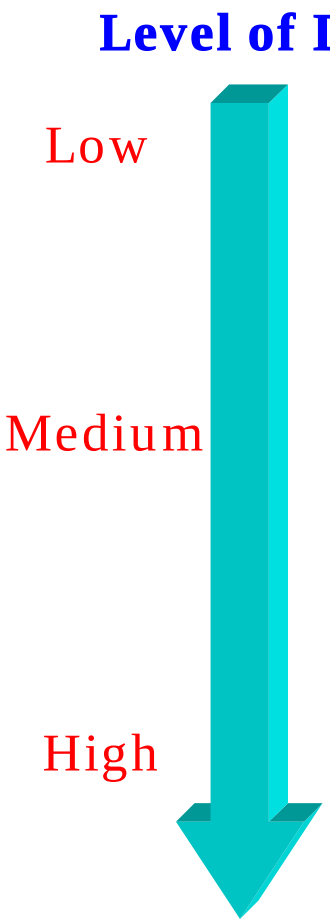

Workload
Source Code Analysis
Object Editor
Object Library
H/W Models
CPU
Network
Cache
Language Scripts
Compiler
Hardware Models
Evaluation Engine
Application Model
Core System
Performance Analysis
On the Fly
Outputs

Source Code Analysis



- Use SUIF to extract computational and memory resources
- Profiler, and manual specification, of looping & branching

Why SUIF?



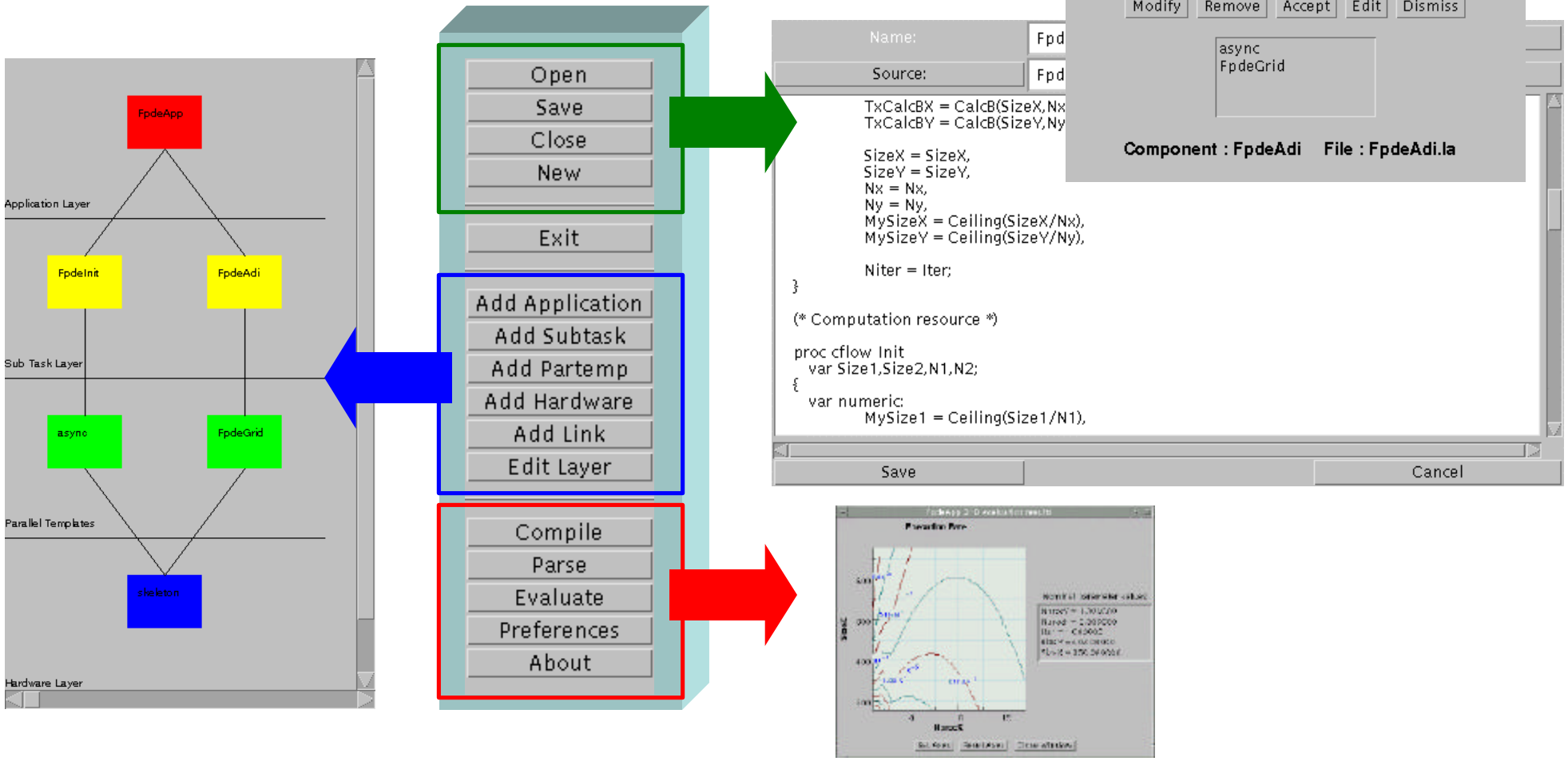
- High Level Code Characterisation
  - Automation, architecture neutral
  - Inaccurate, language specific
- Intermediate Code Characterisation
  - Language and architecture independent
  - Accurate, easy to automate, optimisation
  - Support for C, Fortran, C++ and Java
- Instruction Code Characterisation
  - Accurate, language neutral
  - Difficult to automate, system dependent

Manual Workload Definition

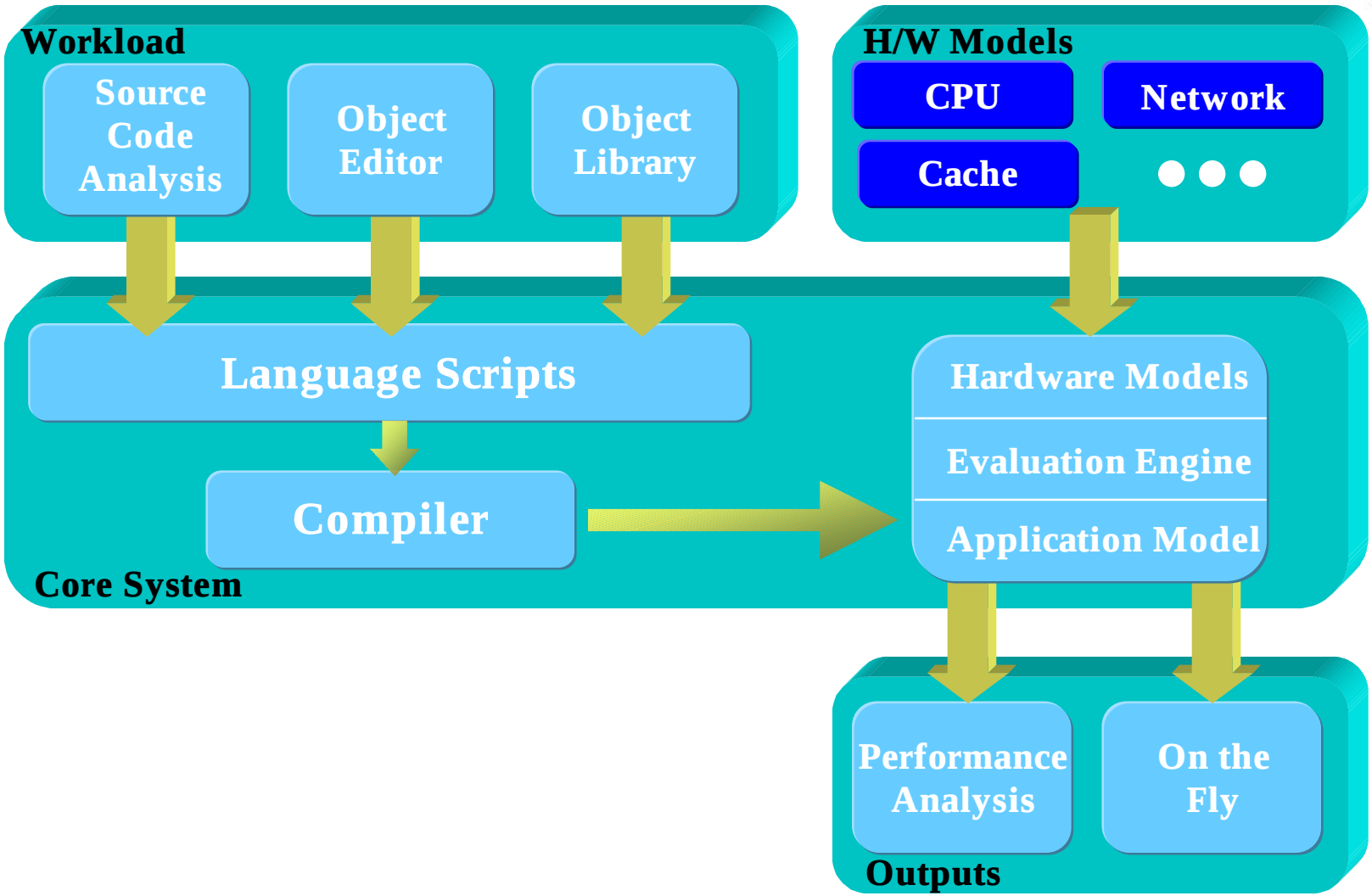
THE UNIVERSITY OF WARWICK

- When?
  - Source code not available (e.g. design stage)
  - Computational code extracted from sequential code
  - Experimentation in a high level of abstaction
- Tools
  - Workload Definition Environment
  - Libraries of objects

Workload Definition Environment

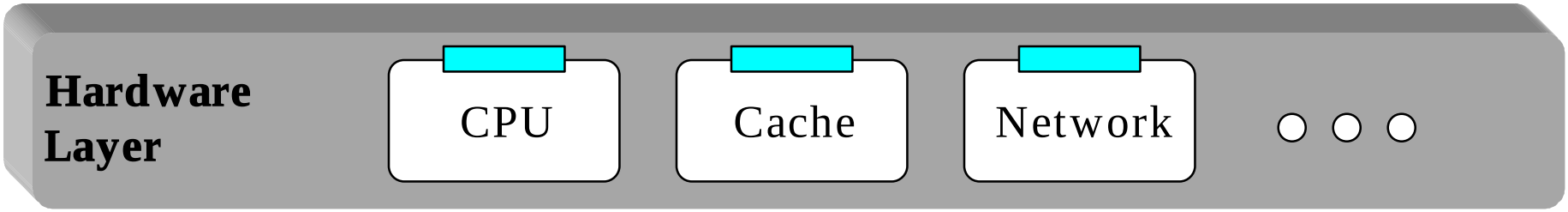


Hardware Models



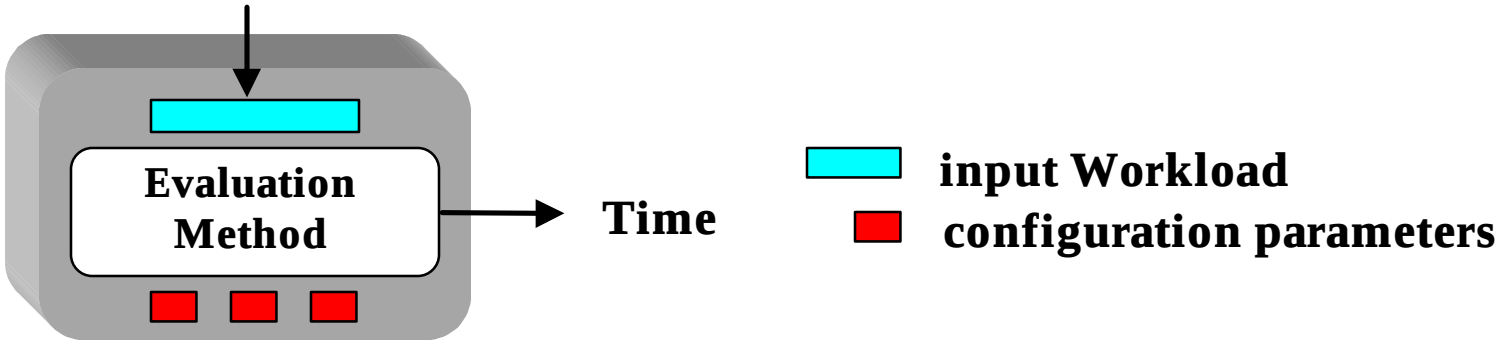
Hardware Models

- Individual models for system components
  - Modularity
  - Multiple levels of abstraction
- Workload interface
- Composite models in hardware layer



Component Models

- Each component model contains:
  - interface to workload (input)
  - configuration parameters
  - evaluation method (output time prediction)

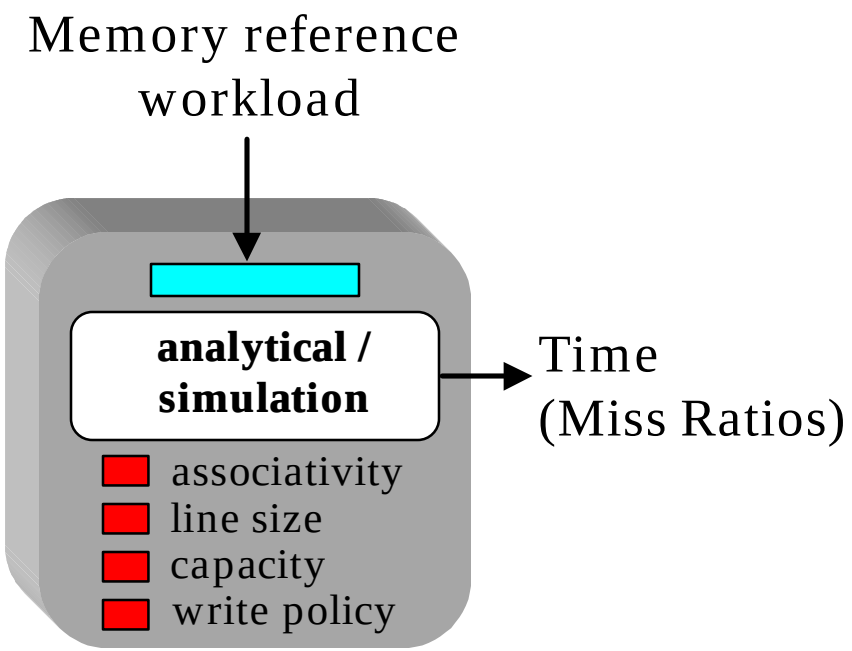


- Encapsulate detail
- Plug In

Type of Models

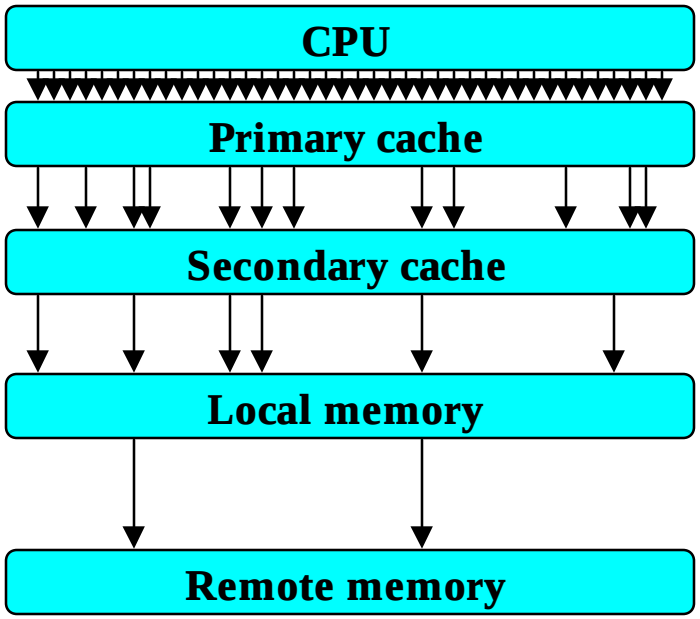
- Focused on generic models
  - common characteristics and heuristics
- Analytical based
- Models include:
  - CPU
    - » C, Fortran (Language construct costs)
    - » SUIF (Intermediate format costing)
  - Network
    - » MPI, PVM (message passing interfaces)
  - Cache
    - » Direct mapped, Set associative, L1, L2

Example: Cache Model



- Hybrid
- Multi-level
  - primary, secondary
- nested loop structures
- Fast
  - orders quicker than sim.
- Accurate
  - <10% error on Miss ratio

Cache Model - approach



- Multi-level
- Each level acts as a filter
- workload refined between levels

Cache - Workload

- Workload (extracted from code)

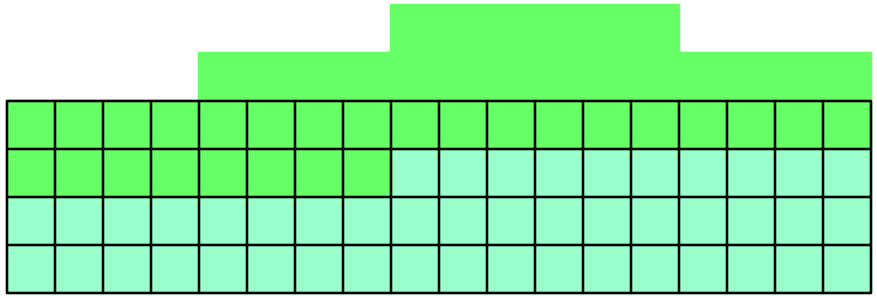
```
DO I = 0, N-1
  DO J = 0, N-1
    Z(J, I) = 0.0
    DO K = 0, N-1
      Z(J,I) += X(K,I)*Y(J,K)
    ENDDO
  ENDDO
ENDDO
```

→

```
array X:double[N,N] at 0
array Y:double[N,N] next
array Z:double[N,N] next
loop 0, N-1
  loop 0, N-1
    Z[loop(2), loop(1)]:w
    loop 0, N-1
      X[loop(3), loop(1)]
      Y[loop(2), loop(3)]
      Z[loop(2), loop(1)]:rw
```

- Refined using cache 'Footprints'

```
DO J = 0, M-1
   DO I = 0, N-1
      X(I) = Y(I)
      ...
```



Hardware Model Configuration Language (HMCL)

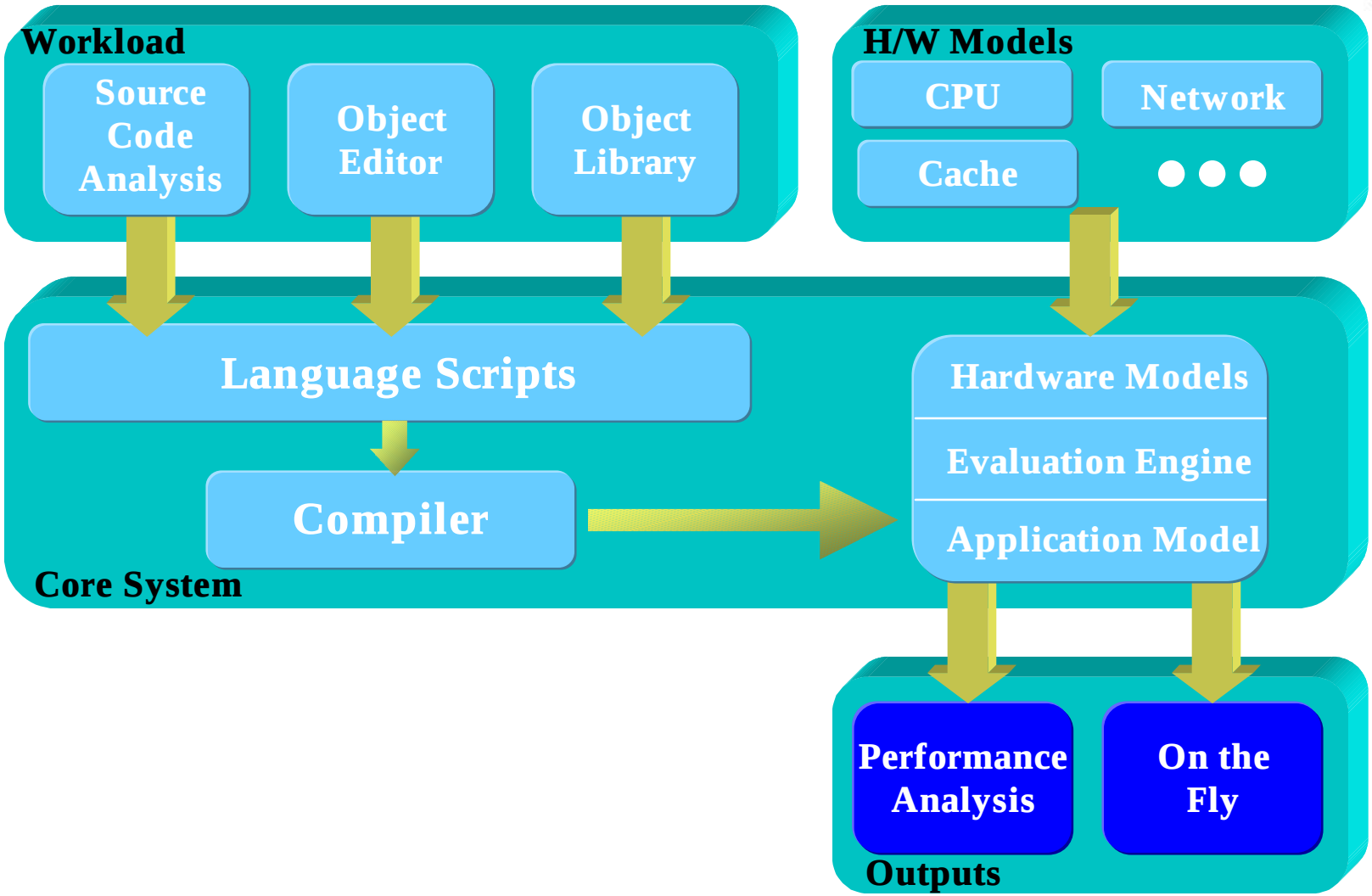
- HMCL specifies target system
- Combination of:
  - component performance models
  - configuration parameters
  - connectivity
- Heterogeneous
- Possible to consider dynamic changes

Example HMCL description

```
component   computer     Sun4 {
        cpu_clc {
                IASG = 2.2,
                DADD = 4.1,
                ...
        }
        cache_l1 {
                Capacity = 16384,
                Assoc = 2, LSize = 64;
        }
        ...
}

component   network      Ethernet {
        PVM {
                TxASend(l) = 539 + 23.l,
                ...
        }
}

Sun4 ClusterA[32];
Ethernet     ANet;

connect      ClusterA[1:32} to ANet;
```

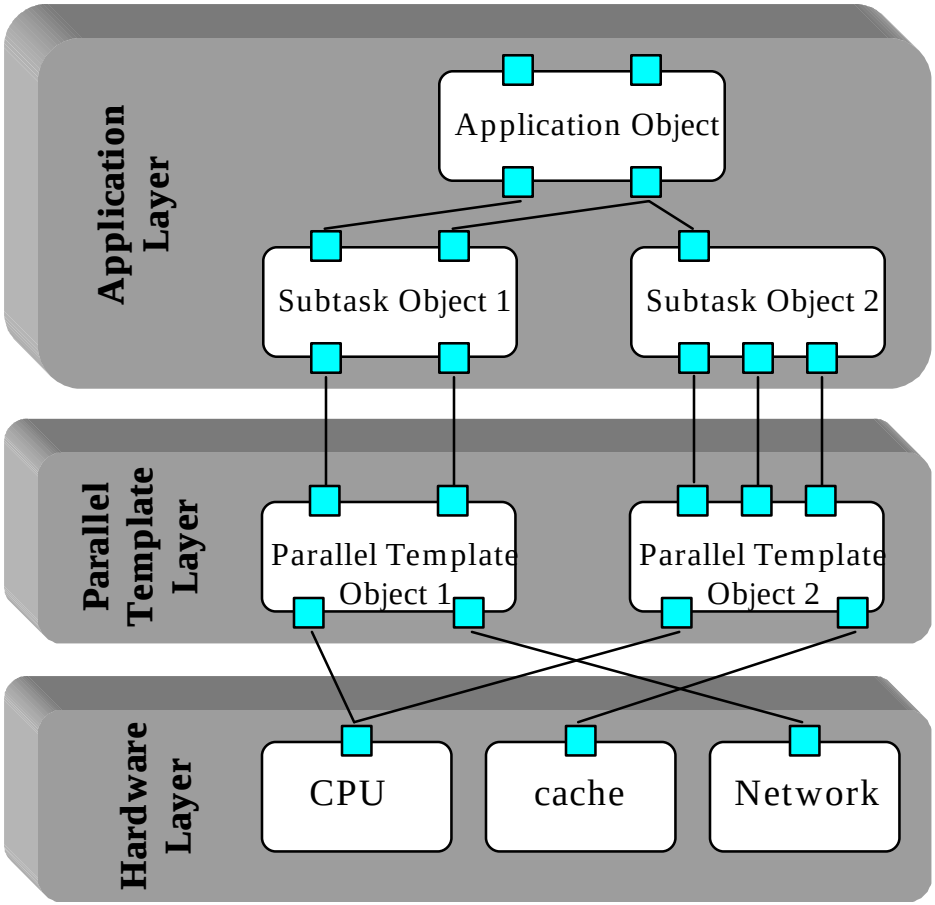
- Two components:
  - Sun4,
  - Ethernet
- performance models:
  - cpu_clc,
  - cache_l1,
  - PVM
- 32 nodes on Ethernet

PACE - Outputs

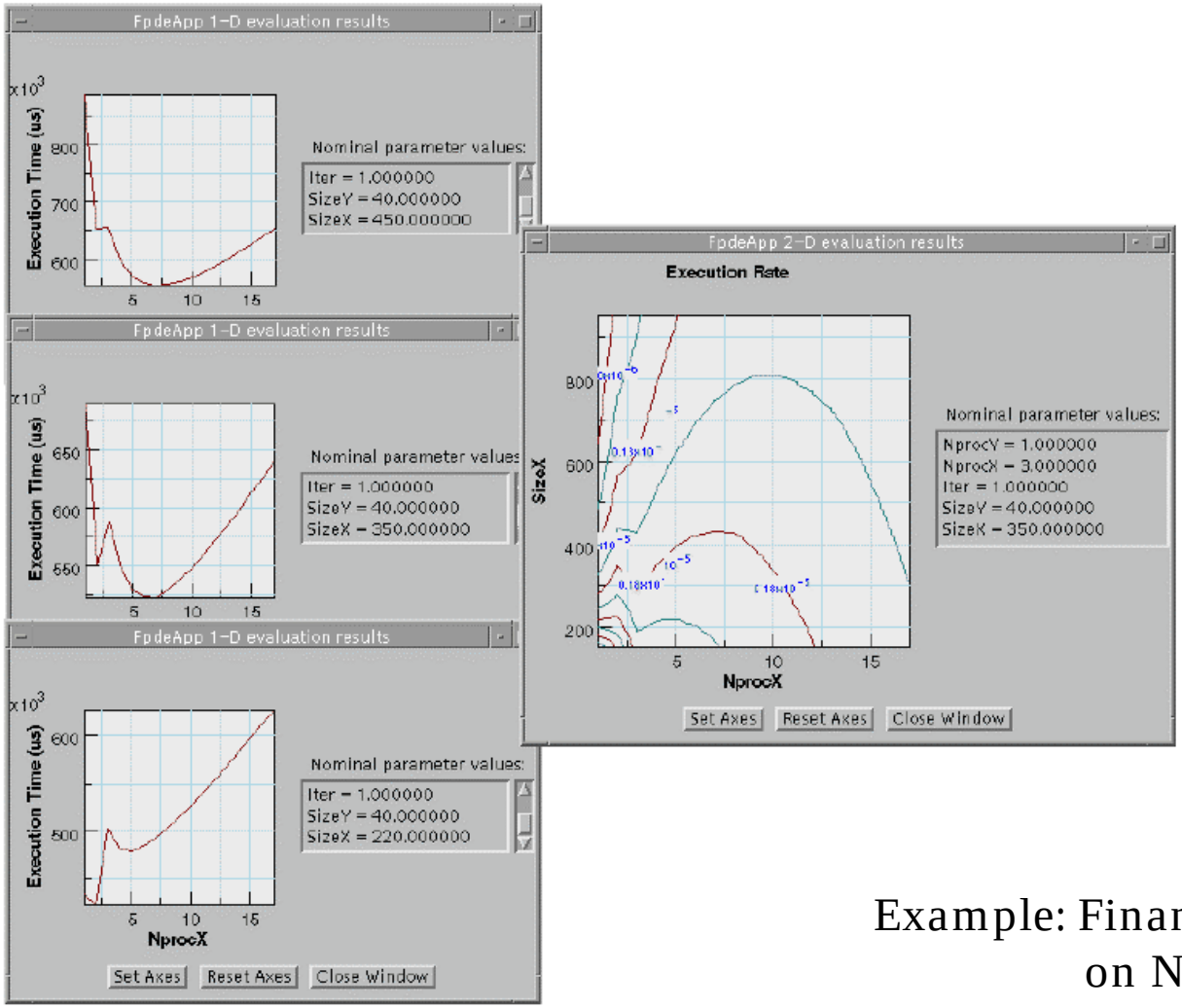


Performance Model Decomposition

- Parameters include:
  - Application
  - System specification
- Input parameters visible to objects (derive internal variables)
- Parameter specification creates scenarios

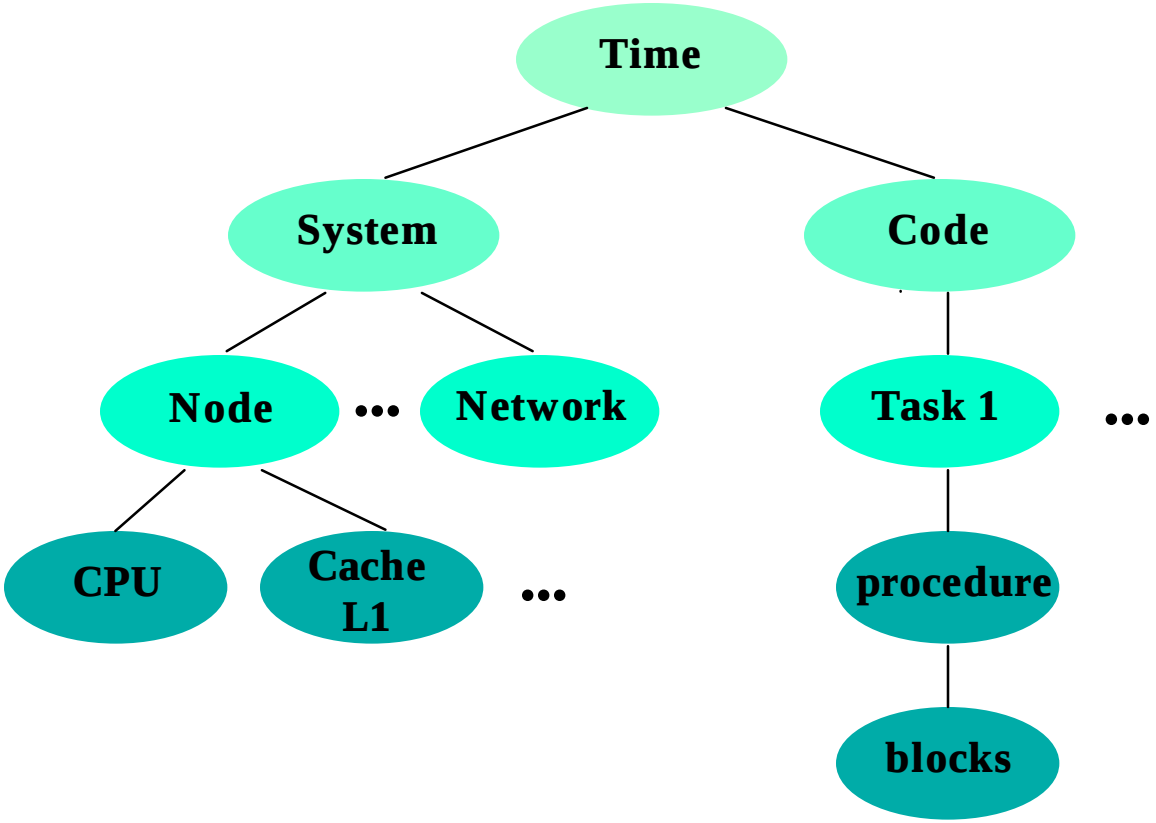


Scalability Analysis



Example: Financial Option
on Network Cluster

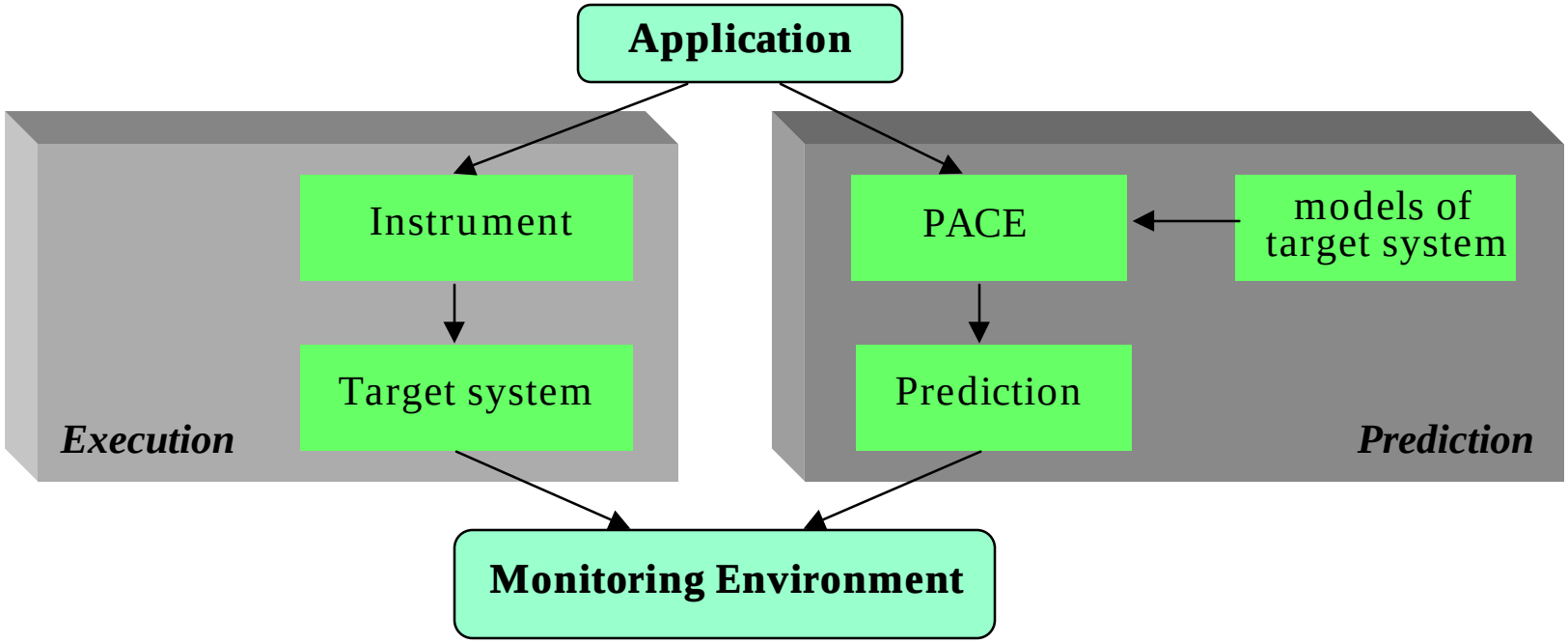
Time Decomposition



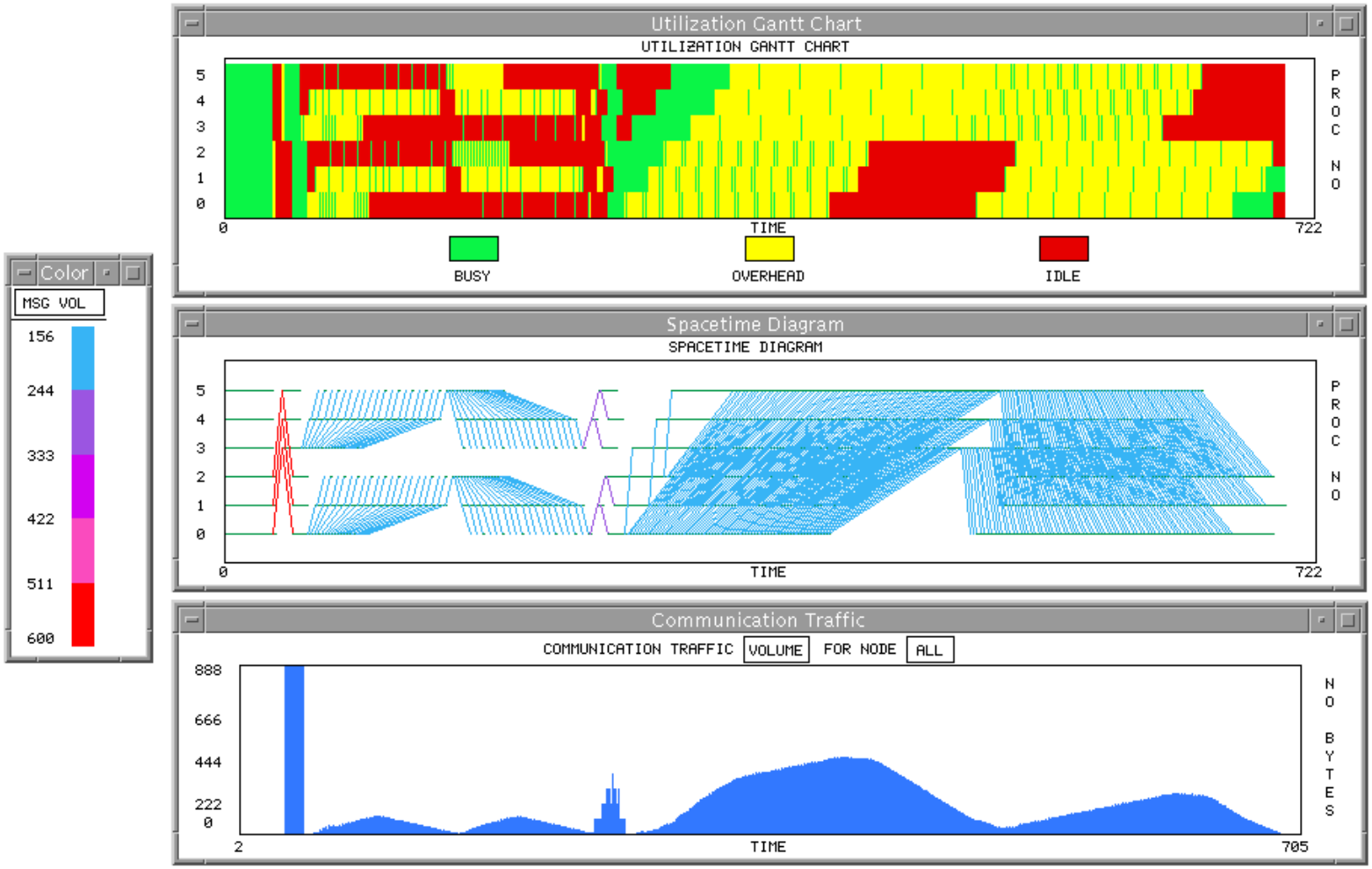
- Hierarchical
- Decomposition into:
  - System
    CPU, cache, network, I/O
  - Application
    sub-tasks, procedures, etc.
- Group Components
  - identify bottlenecks

Predictive Traces

- Generate events from workload information
- Integration with measurement tools for visualization
  - Paragraph (PICL format, ORNL)
  - Pablo (SDDF format, UIUC)



Example visualization

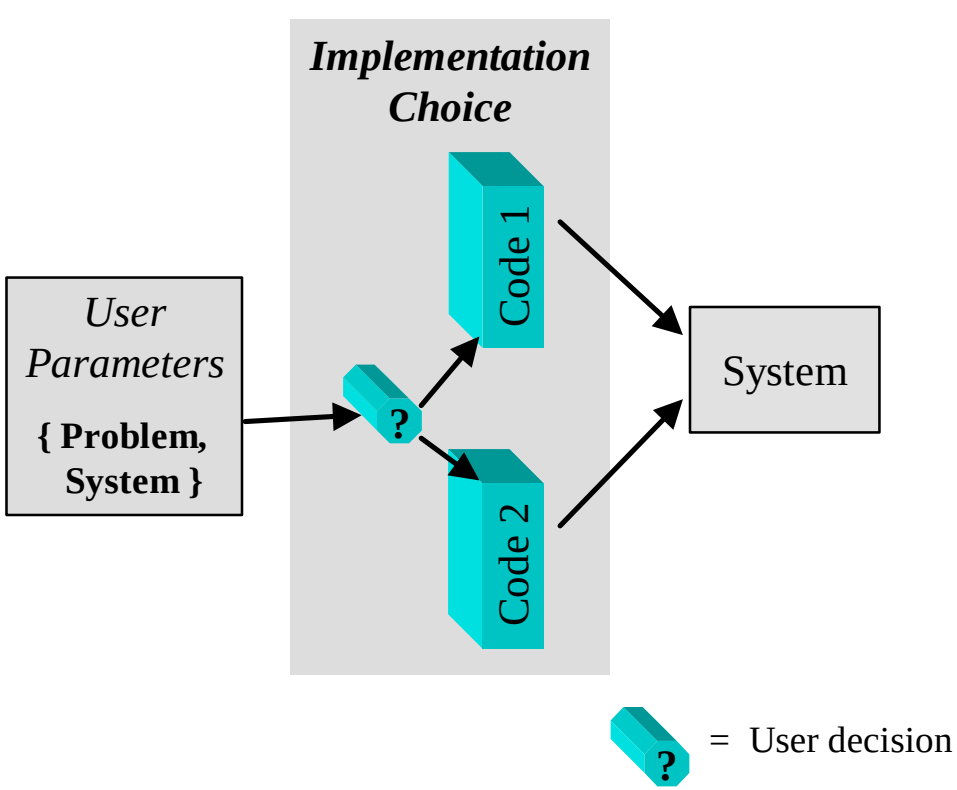


Predictions On-the-fly

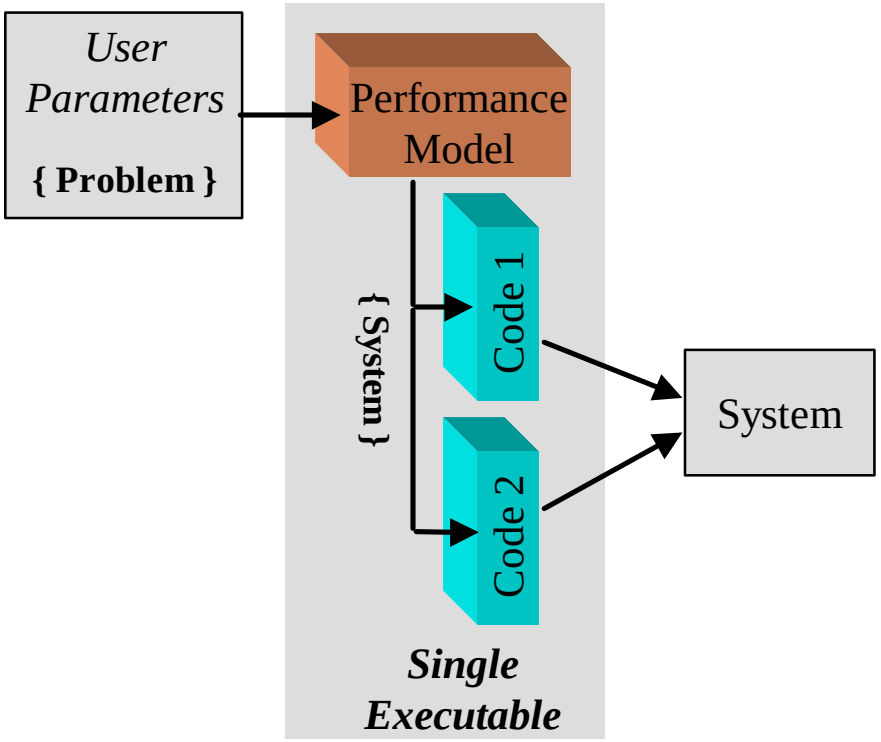
- Fast prediction evaluation
- Pre-execution evaluation
  - Algorithmic choices
  - system configuration
- Extend to:
  - multi-task Scheduling
  - dynamic task partitioning (& load balancing)
  - performance instrumentation
- Principal:

  add performance model to application executable

Performance 'stub'



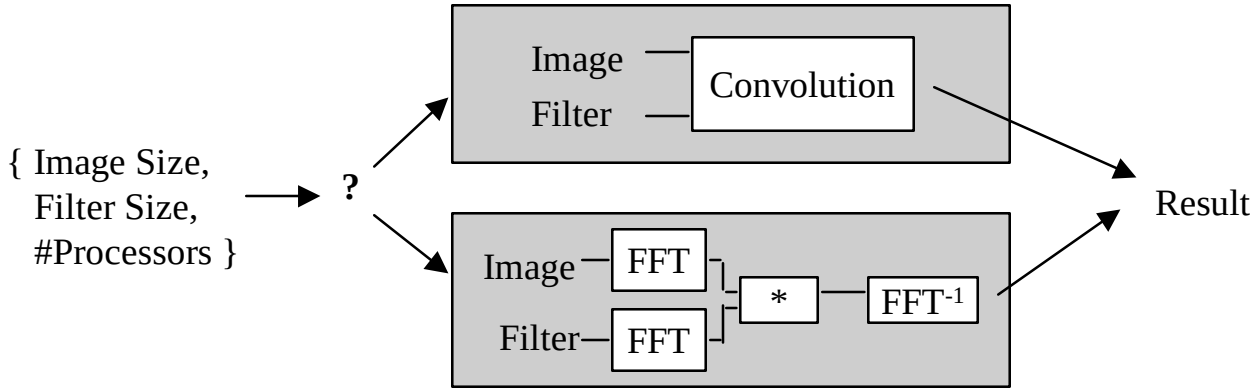
User directed execution



Performance directed execution

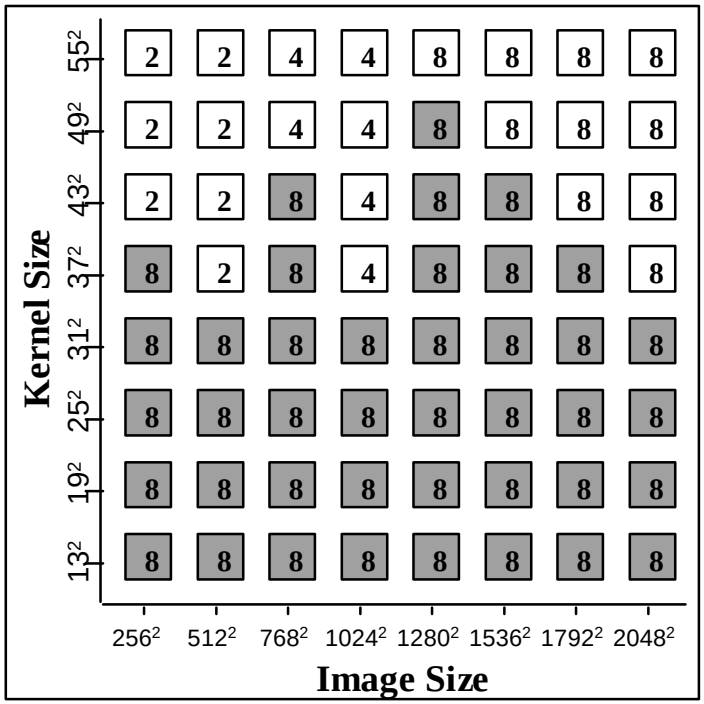
Example (Algorithmic choice)

- Image processing convolution

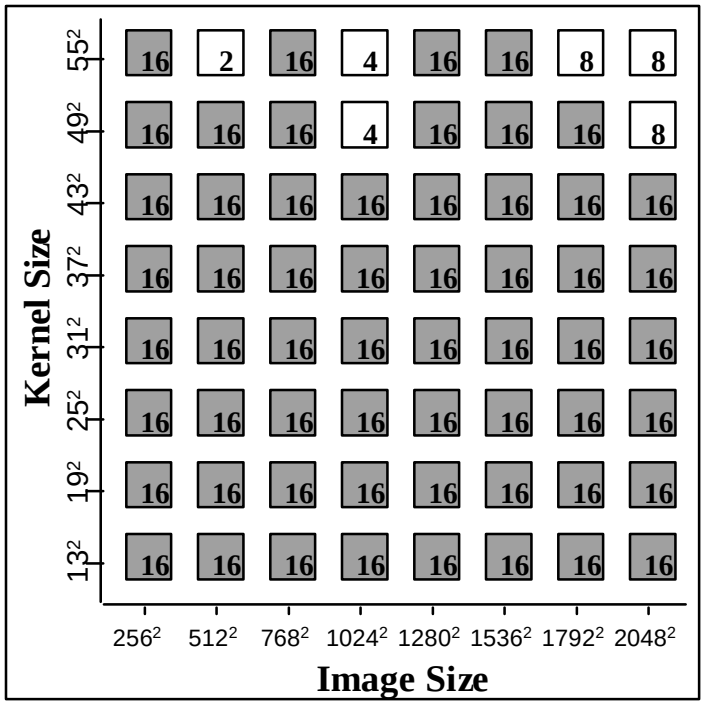


- Problem parameters - image size, filter size
- Parameters determined by performance model:
  - processing method
  - # Processors (System - SUN Ultra Cluster)

Model output



Max: 8 workstations



Max: 16 Workstations

Spatial

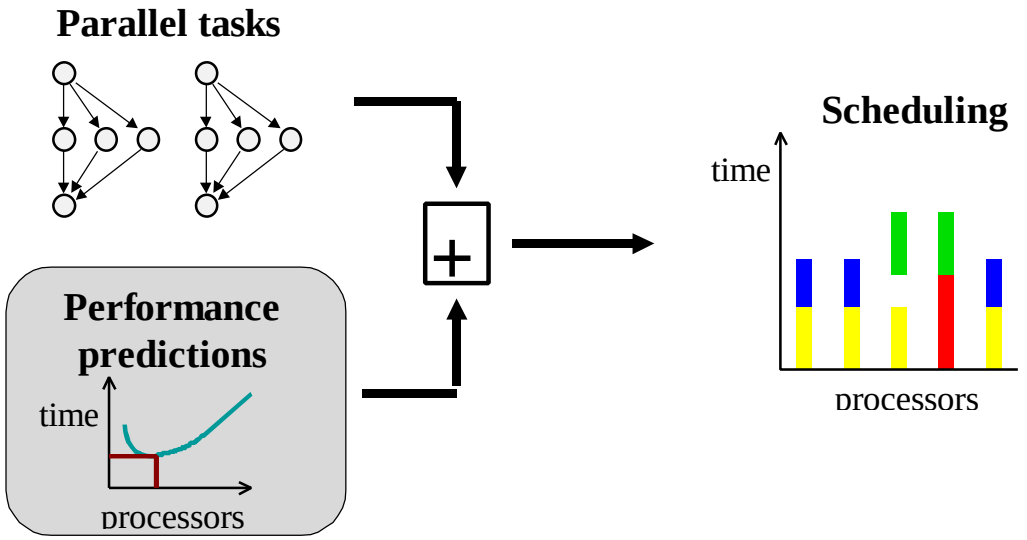
Frequency

(Numbers in each box indicate #workstations to use for given problem parameters)

http://www.dcs.warwick.ac.uk/~hpsg/reference_top.htm

Schedule Optimization

- Use performance model associated with each task
- Pre-execution evaluation
  - time prediction
  - best system mapping
- Optimize schedule



Mutiprocessor Scheduling

THE UNIVERSITY OF WARWICK

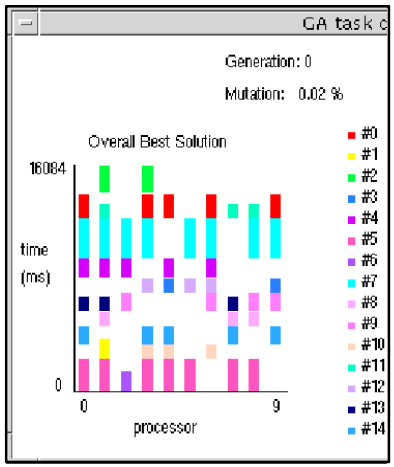
Tasks $\{T_0, T_1, ..., T_{n-1}\}$

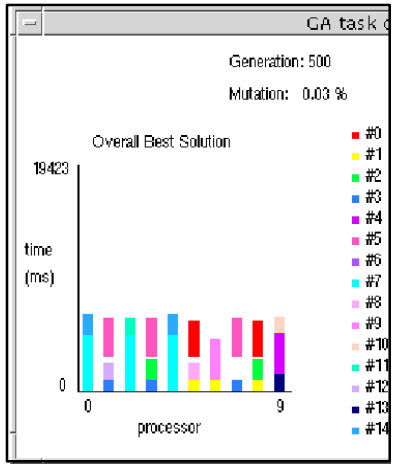
Processors $\{P_0, P_1, ..., P_{m-1}\}$

For each task: $b \in \{0, ..., m-1\}$ Set of processors

t_j t_j Start time, and duration

Goal: minimise the *makespan* $\max_{0 \le j \le n-1} \{t_j + t_j(\| b_j \|)\}$

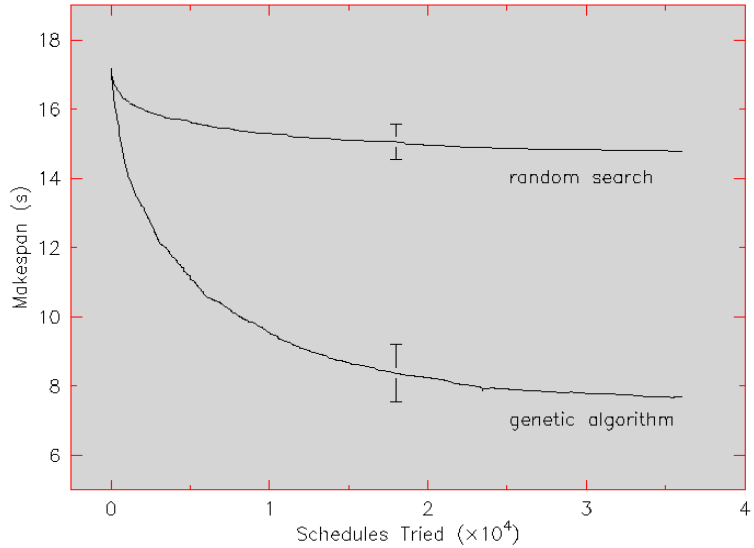




- Time prediction not available in traditional scheduling

Schedule makespan Improvement

- Genetic Algorithm Approach -
  - Evolutionary method
    - » One-time start-up cost
    - » small evolution cost (additional tasks / changing resources)



Example - 15 tasks 10 processors

Applications

THE UNIVERSITY OF WARWICK

- Financial Options
  - » Monte-Carlo Simulation, PDE (UK Banks)
- Image Processing
  - » Real-time image processing (NPL, London),
  - » SAR (NA-Soft),
- Embedded Signal Processing
  - » Signal Processing (Thomson ASM, Nice)
- Graphics
  - » Photo-realistic generation (Mental Images, Berlin)

On-going research

- Shared memory
- Heterogeneous
- Object Orientated
- on-the-fly

http://www.dcs.warwick.ac.uk/~hpsg