



HPEC 2005: Will Software Save Moore's Law?

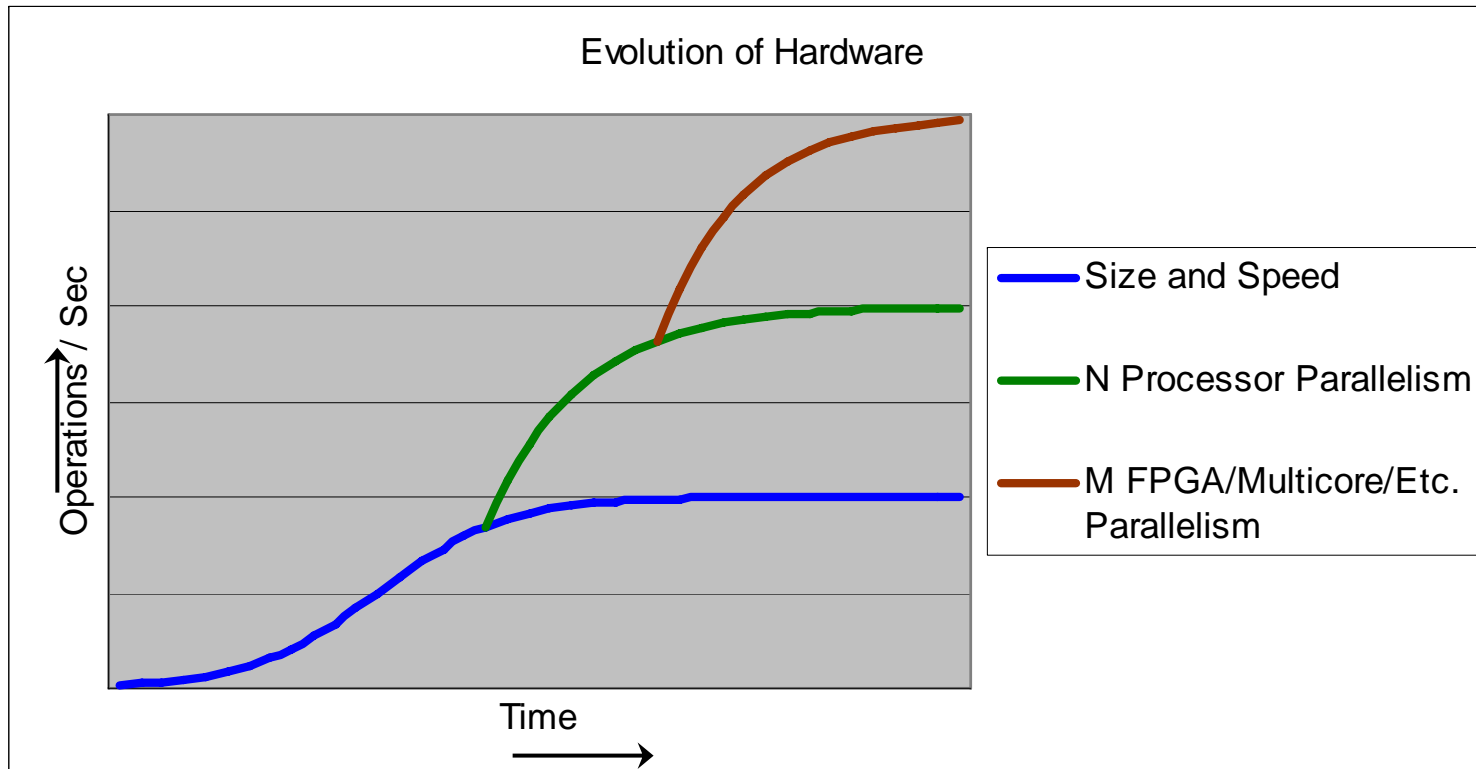
# How Code Generation Can Save Moore's Law

Gedae, Inc.

[www.gedae.com](http://www.gedae.com)

856 - 231- 4458

# Précis: Evolution of Hardware



Processor and Chip Level Parallel Execution  
Extends Moore's Law

# Précis: Programming Must be Geared Toward Parallel Heterogeneous Systems



- Distributed application issues:
  - Distribution and re-distribution
  - Analysis
  - Deadlock avoidance
  - Optimizing to different targets
- Software layers do not adequately address these

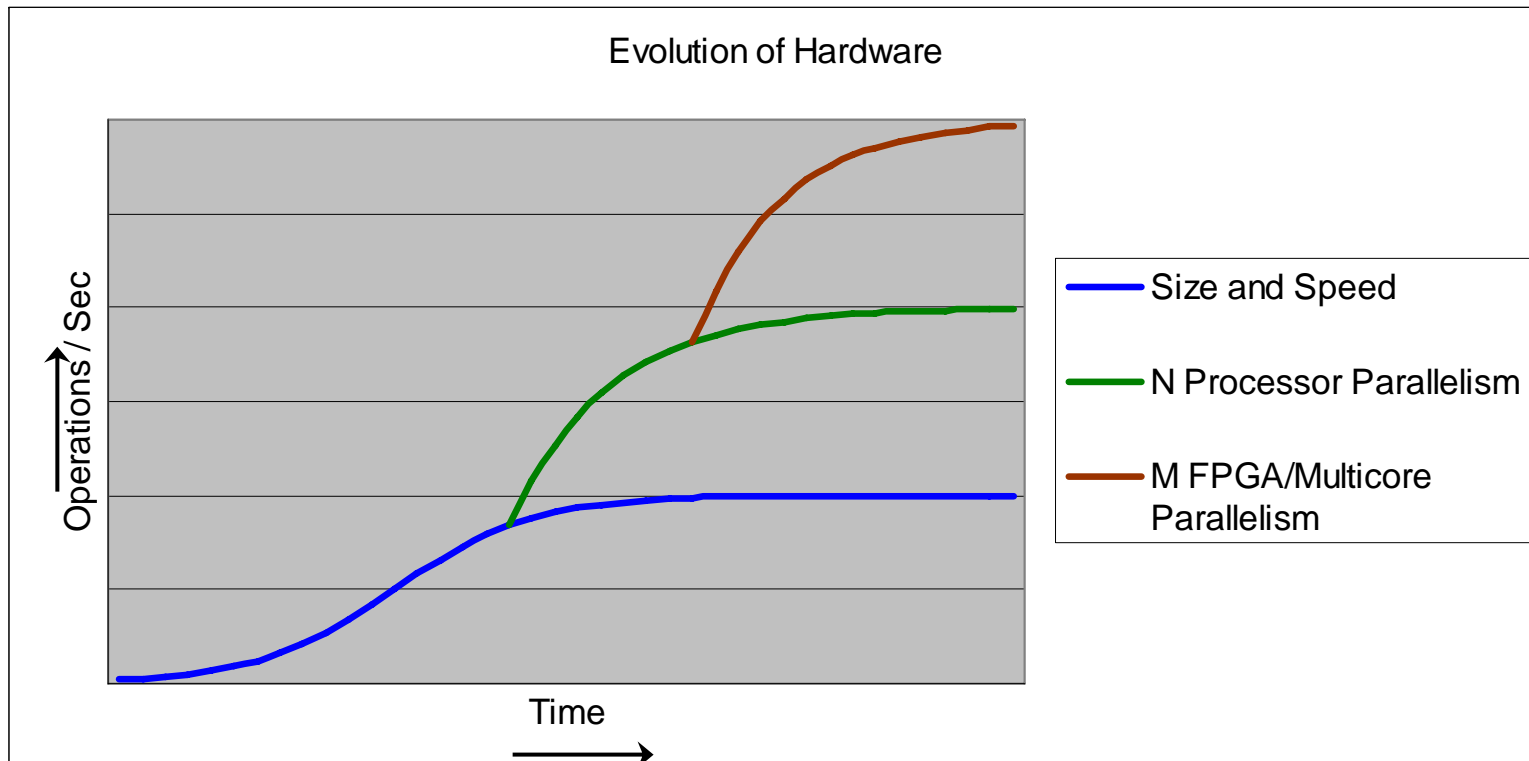
|                   | Software Layers           | Autocoding        |
|-------------------|---------------------------|-------------------|
| New Architectures | Complex to implement      | Easily supported  |
| Efficiency        | Tradeoff with flexibility | Coded close to HW |
| Distribution      | Developer-insured         | Automated         |



## Précis: Savings Moore's Law

- For software to extend Moore's Law, it must be
  - Efficient
  - Robust
  - Built in short development cycles
- Need advanced programming tools that meet these demands while targeting multi-processor systems
- If software is to save Moore's Law, advanced programming tools must be in place to save software development

# Evolution of Hardware



Processor and Chip Level Parallel Execution  
Extends Moore's Law



# Programming Must be Geared Toward Parallel Heterogeneous Systems

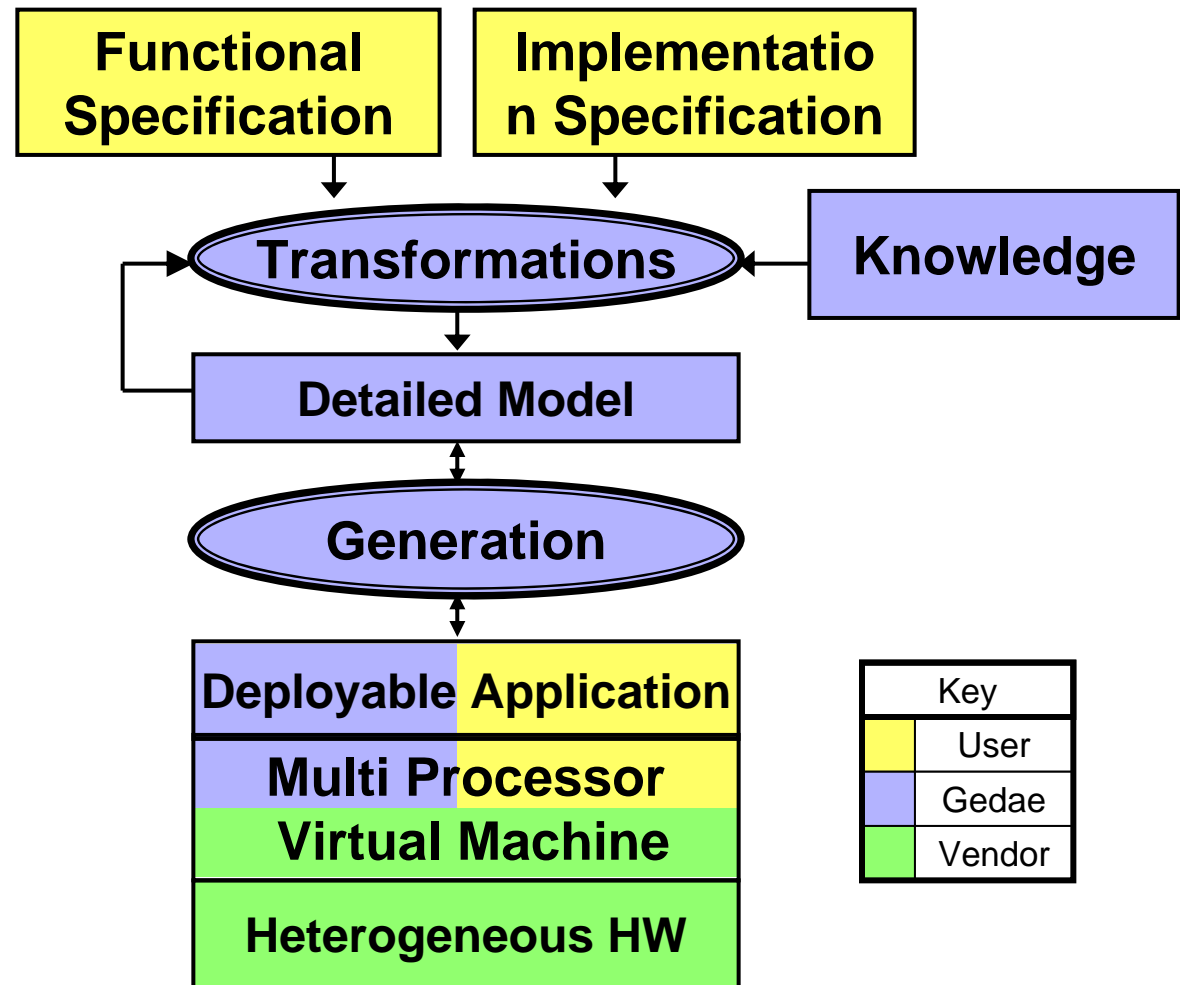
- What's needed to program distributed applications:
  - Distribution and re-distribution
  - Analysis
  - Deadlock avoidance
  - Optimizing to different targets
- Software layers do not adequately address these

|                   | Software Layers           | Autocoding        |
|-------------------|---------------------------|-------------------|
| New Architectures | Complex to implement      | Easily supported  |
| Efficiency        | Tradeoff with flexibility | Coded close to HW |
| Distribution      | Developer-insured         | Automated         |

# Separation of Functionality and Implementation

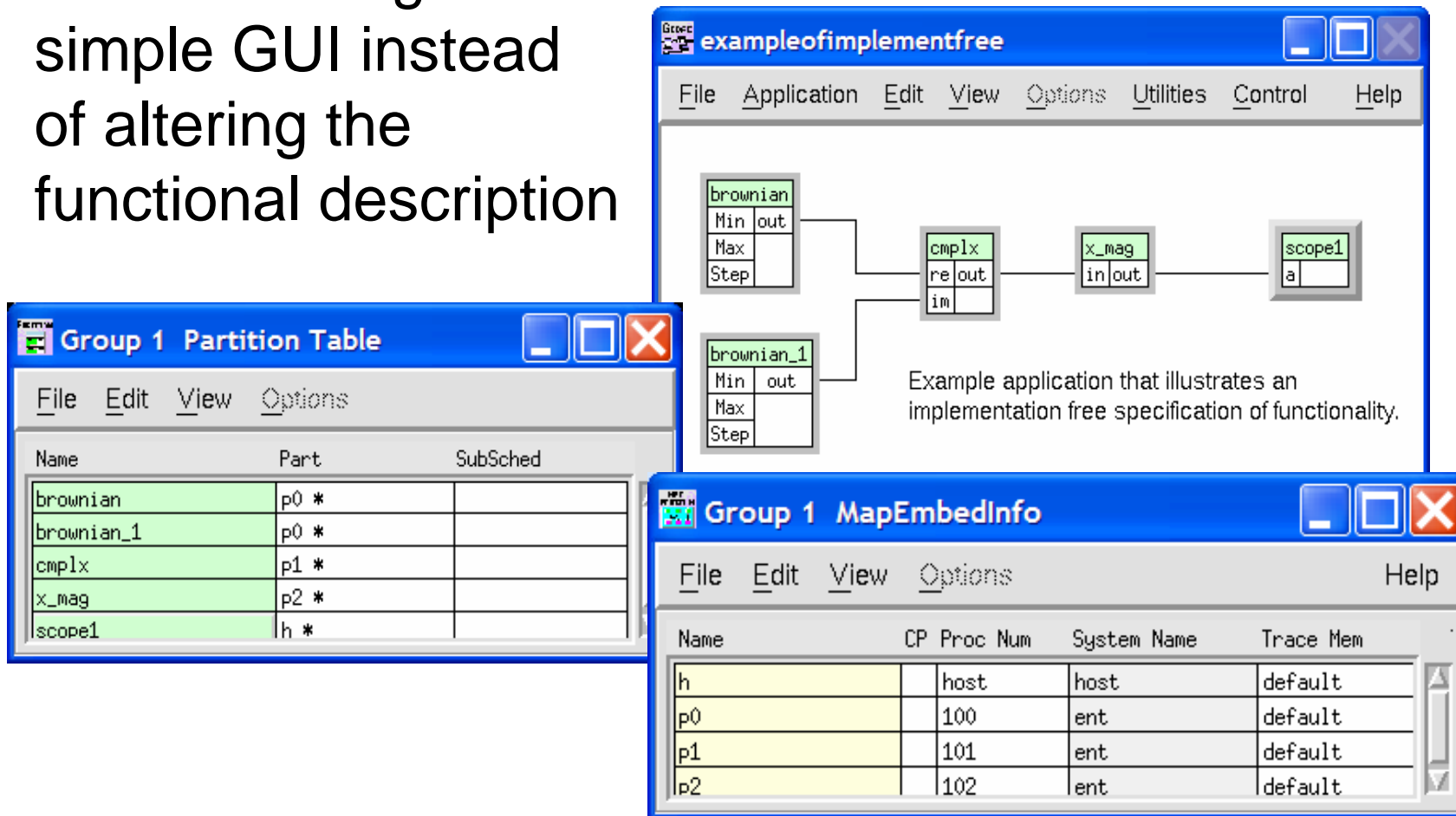


- Functional description should not change when going from one implementation to another
- Knowledge based code generation produces an efficient implementation



# Example: Partitioning and Mapping

- Handle through simple GUI instead of altering the functional description



The screenshot displays three windows from the Gedae application:

- exampleofimplementfree**: A window showing a functional diagram with components: `brownian` (Min, Max, Step), `brownian_1` (Min, Max, Step), `cmplx` (re, im), `x_mag` (in, out), and `scope1` (a). A text box below the diagram reads: "Example application that illustrates an implementation free specification of functionality."
- Group 1 Partition Table**: A window with a table showing the partitioning of components.
 

| Name       | Part | SubSched |
|------------|------|----------|
| brownian   | p0 * |          |
| brownian_1 | p0 * |          |
| cmplx      | p1 * |          |
| x_mag      | p2 * |          |
| scope1     | h *  |          |
- Group 1 MapEmbedInfo**: A window with a table showing the mapping of components to hardware resources.
 

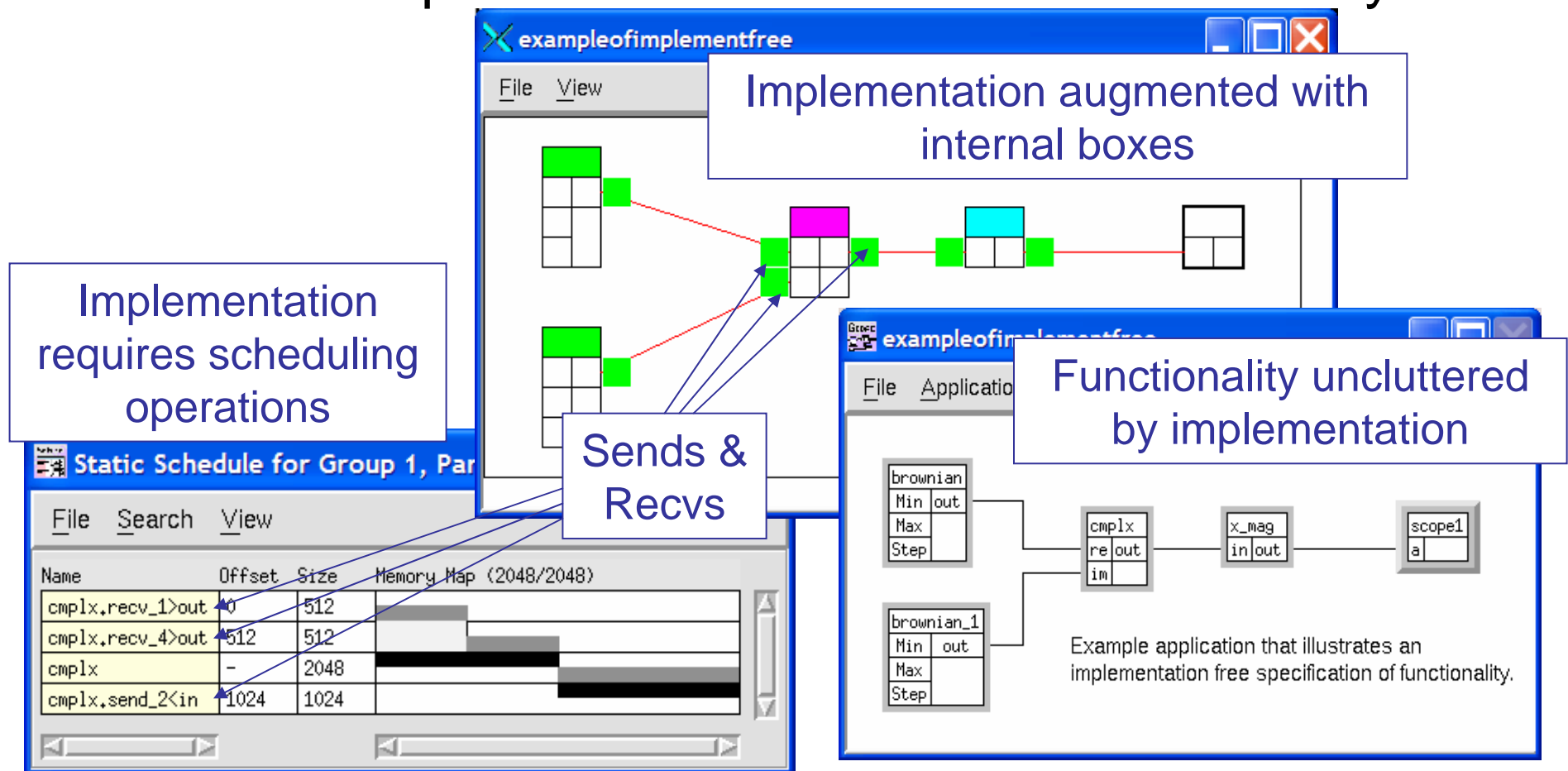
| Name | CP | Proc Num | System Name | Trace Mem |
|------|----|----------|-------------|-----------|
| h    |    | host     | host        | default   |
| p0   |    | 100      | ent         | default   |
| p1   |    | 101      | ent         | default   |
| p2   |    | 102      | ent         | default   |



# Transformations Create the Implementation

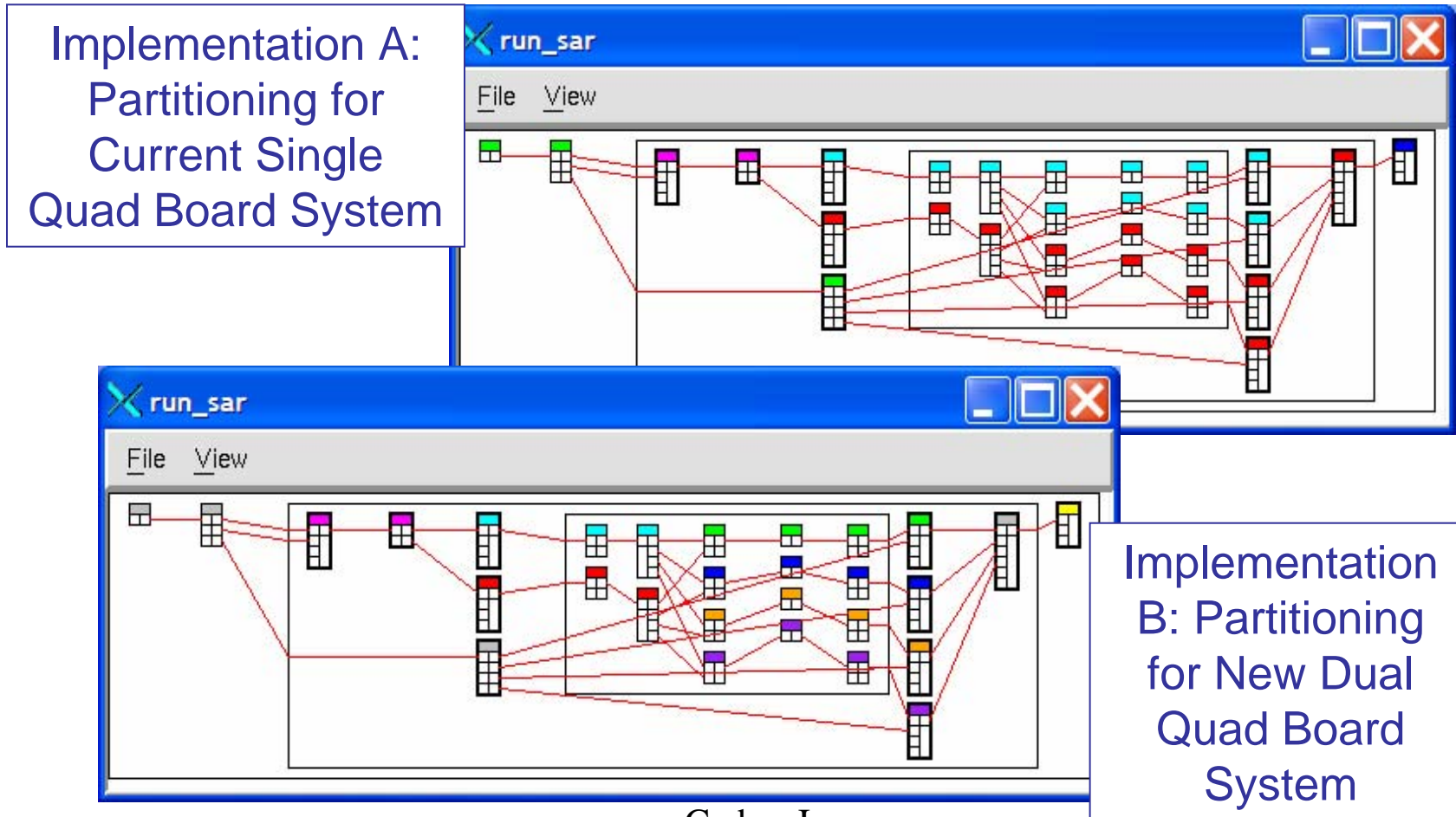


- Gedae uses its knowledge of the virtual machine to create an implementation around the functionality



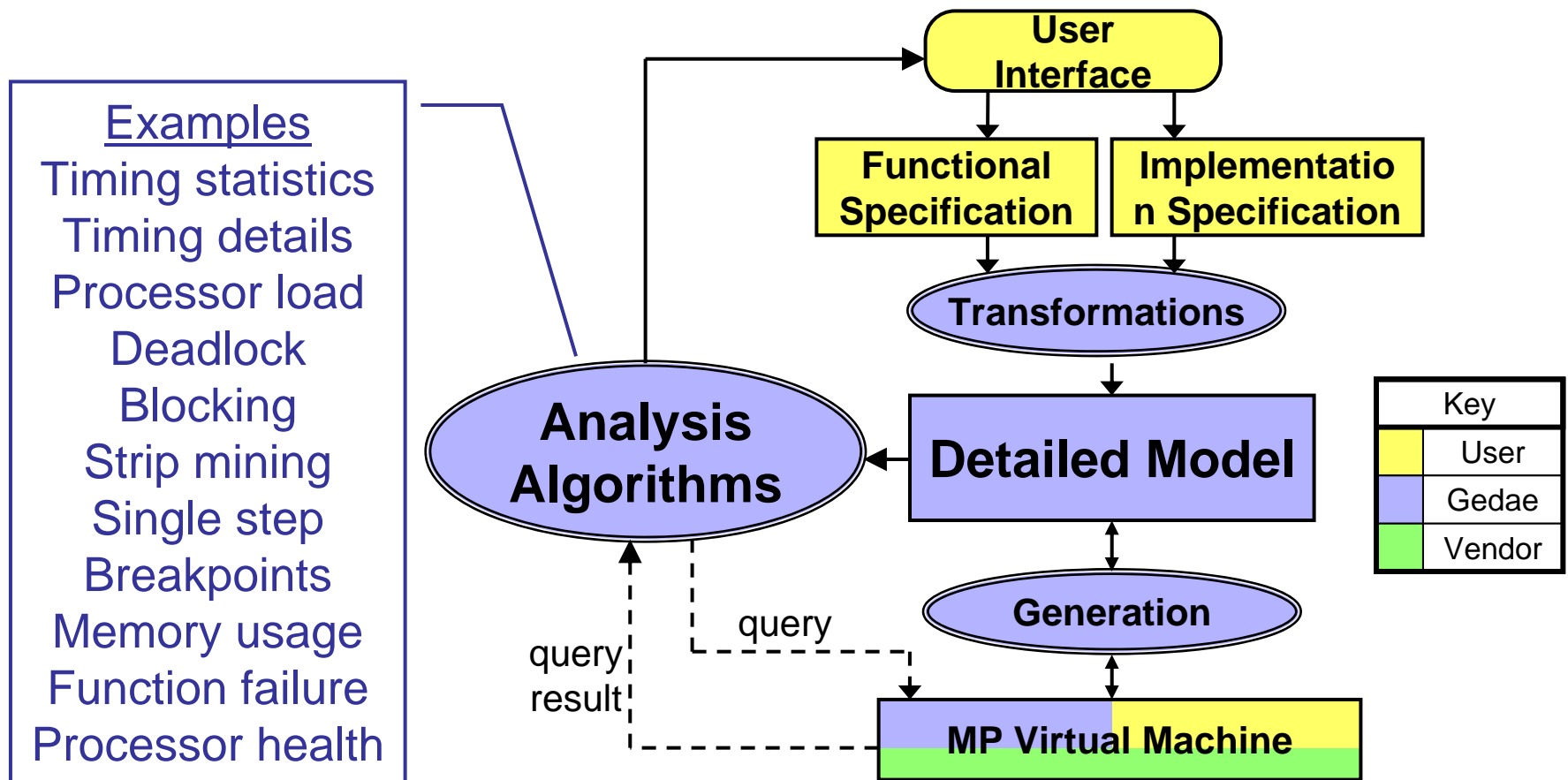
# Software Obsolescence and Portability

- Can easily port to new hardware when the functional description is free of implementation detail



# The Software Model

- Maintaining an internal model of the application provides a parallel debugging & analysis environment



# Analysis of Implementation and Execution: Diagnosing Problems



- Deadlock due to functional error identified by analysis algorithms

The screenshot displays the Gedae software interface with several key components:

- Implementation Window:** Shows a data flow graph with nodes like `brownian`, `cmplx`, `x_dcopy`, `x_rdiv`, `x_mag_1`, `brownian_1`, `x_mag`, and `runningAvg`. The `x_rdiv` node is highlighted in red and labeled "Blocked", while the `runningAvg` node is highlighted in green and labeled "Starved".
- Execution details Window:** Displays a trace table for "exampleofdeadlock" with a Gantt chart showing the execution timeline of various schedules (Schedule 1 and Schedule 2) and their associated tasks.
- Message Window:** Contains the following text:
 

```

      ----- Deadlocked Loop:
      Blocked Schedule 1  (1,0,0,0)
      2: x_rdiv<a          (1,0,1,1,0,1)
      Starved Schedule 2  (0,0,0,1)
      1: x_rdiv<b          (0,0,0,1,1,0)

      Note:
      Each blocked schedule in deadlock loop is followed by blocked outputs.
      Starved schedules are followed by starved inputs
      
```
- Functionality Window:** Shows a block diagram of the application with explanatory text:
 

Inserts queue for illustration purposes.

Running average box artificially drops an occasional token to illustration data flow deadlock.

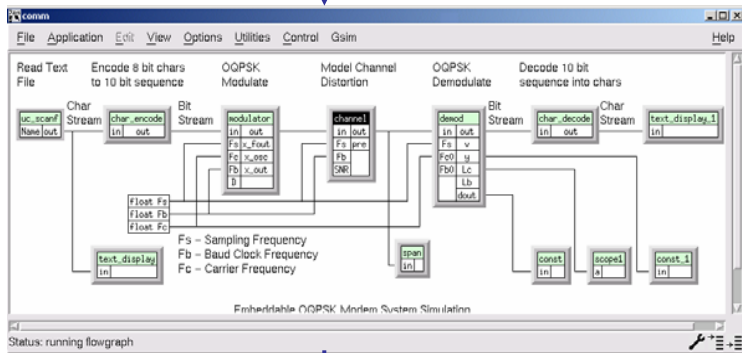
Example application that artificially generates data flow deadlock to illustrate debugging tools.

# Analysis of Implementation and Execution: Optimizing Performance



**Build Application**

Try new algorithms



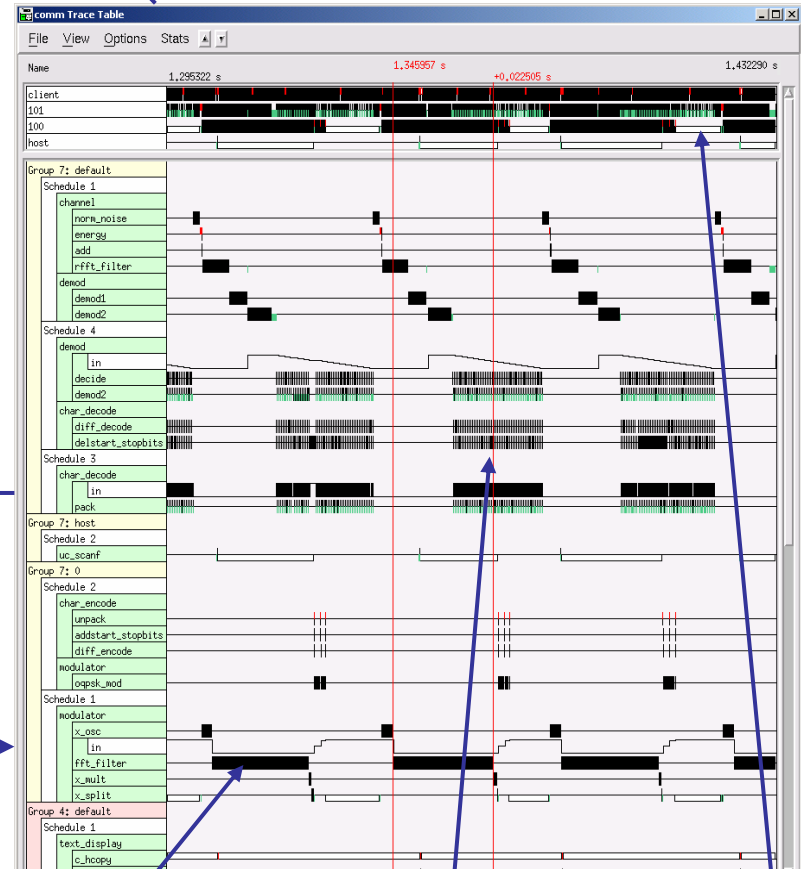
**Test & Optimize**

Try new implementation

Group 7 Partition Table

| Name              | Part    | SubSched |
|-------------------|---------|----------|
| uc_scanf          | host    |          |
| char_encode       | 0       |          |
| diff_encode       | 0       |          |
| addstart_stopbits | 0       |          |
| unpack            | 0       |          |
| modulator         | 0       |          |
| x_osc             | 0       |          |
| x_split           | 0       |          |
| x_mult            | 0       |          |
| fft_filter        | 0       |          |
| oqpsk_mod         | 0       |          |
| channel           | default |          |
| demod             | default |          |
| char_decode       | default |          |

**View Execution**



Slow primitive

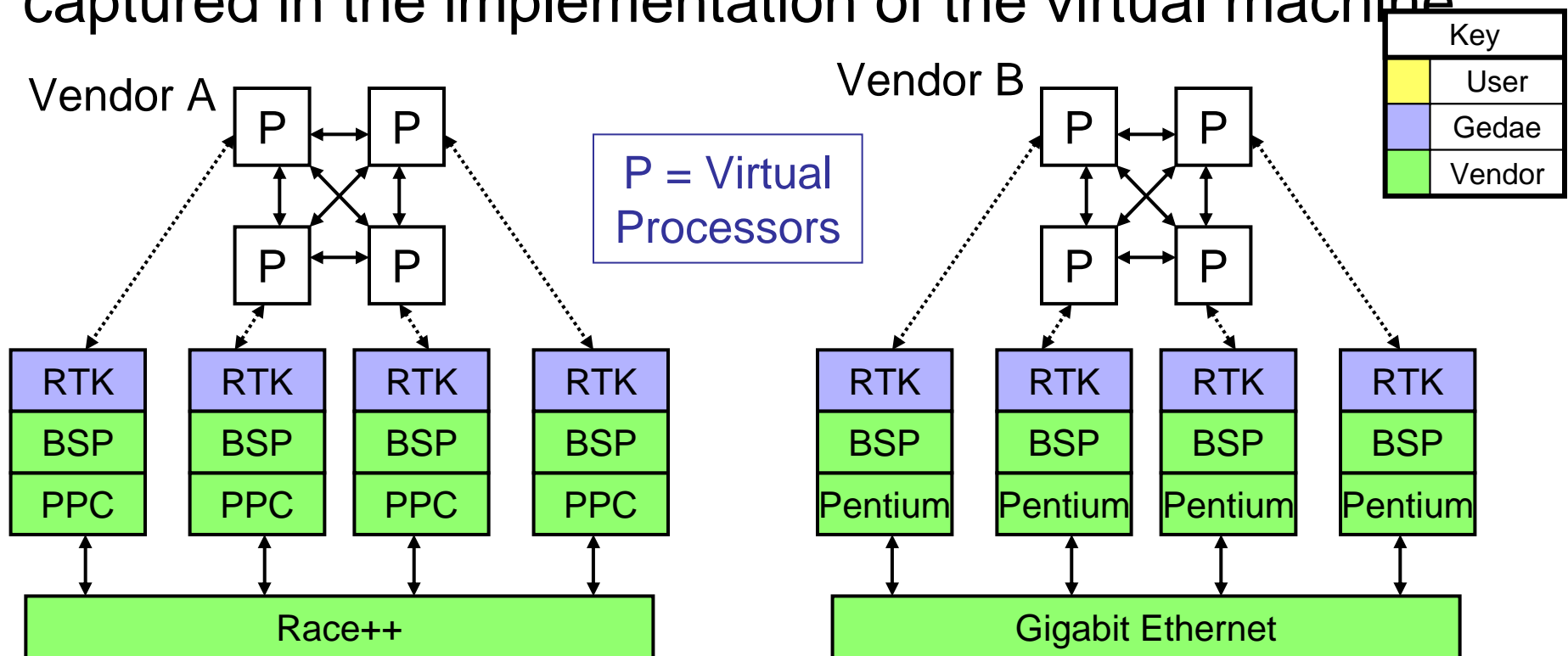
Granularity too low

Not load balanced

# Portability Through a Parameterized Virtual Machine



- Virtual machine is N fully connected processors, each running a Runtime Kernel (RTK)
- Target-specific details (e.g., vendor's BSP) are captured in the implementation of the virtual machine

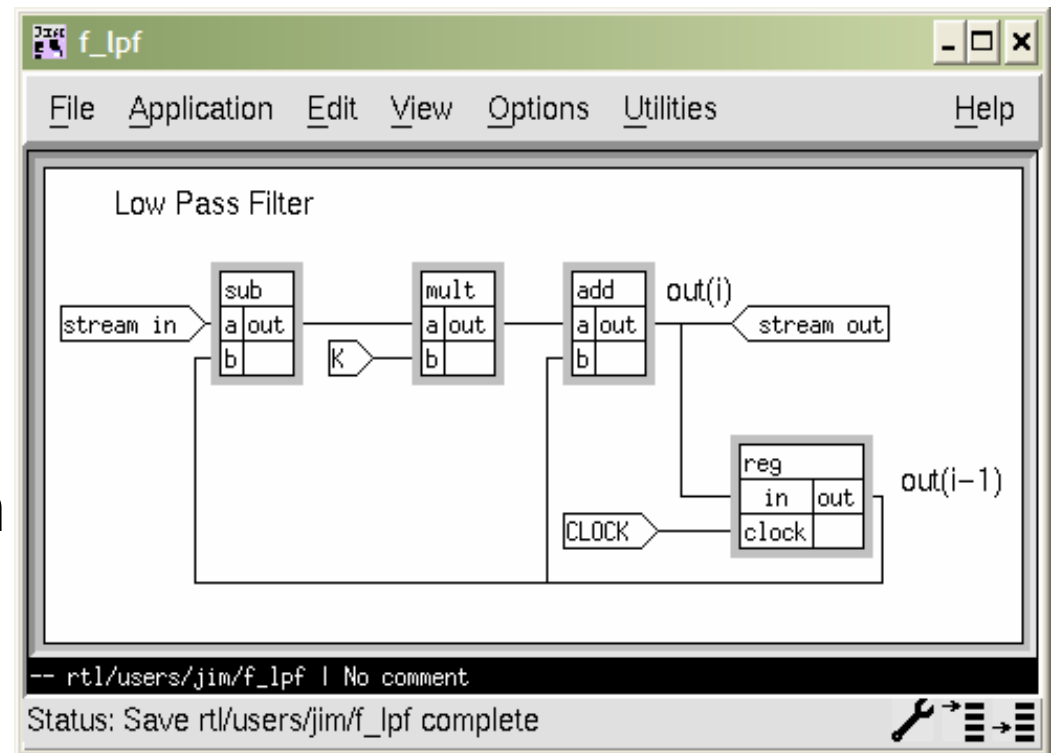


# Targeting Alternate Architectures



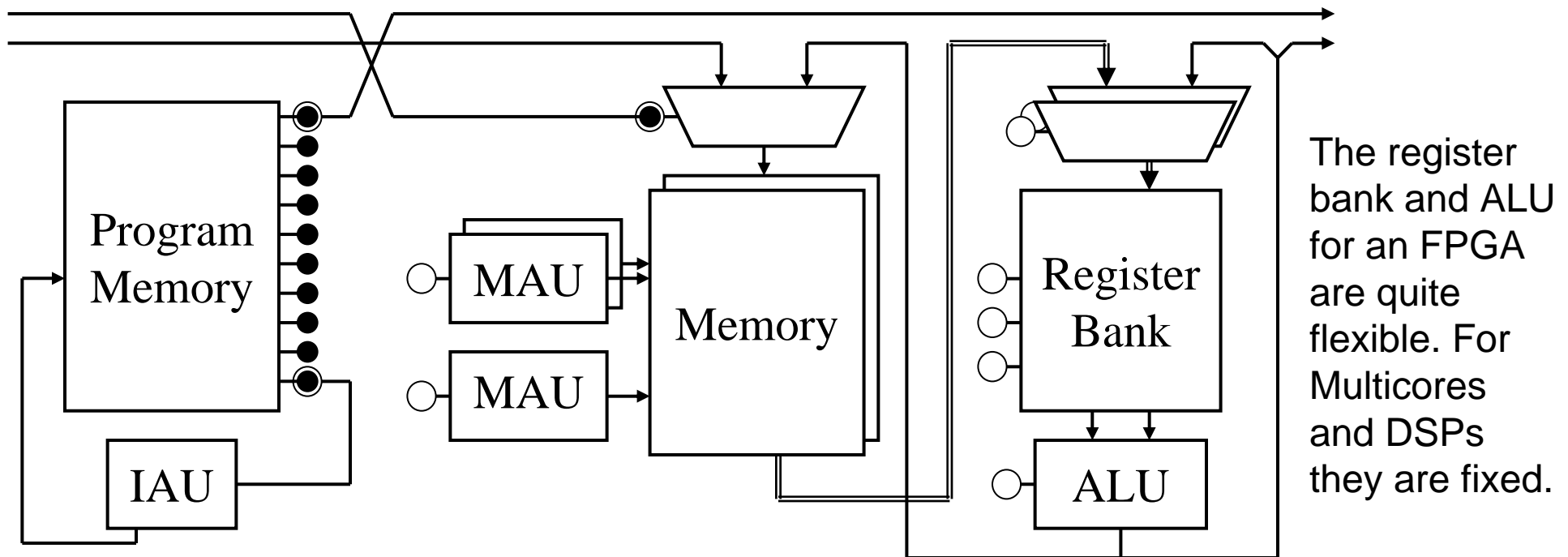
- FPGAs – Not all processors use C code
  - Need Language Support Package to generate VHDL, Assembly, etc. for those processors
- Multicore – Not all processors have enough memory for the RTK
  - Need to generate lightweight processes for those processors

## Single Sample Language



$$\text{out}(i) = K * (\text{in}(i) - \text{out}(i-1)) + \text{out}(i-1)$$

# Taking the Virtual Machine to the Processor Level



- Create virtual processor that emulates target architecture
- Map single sample graph to virtual processor
- Generate low level code through LSP
- Reduce dependency on compilers & vector libraries