Performance Visualization

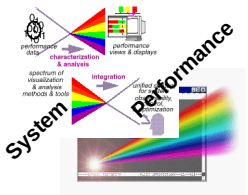
Usability and Reusability

Diane T. Rover

Department of Electrical & Computer Engineering

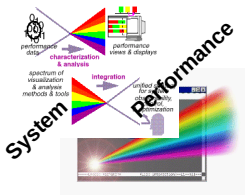
www.egr.msu.edu/~rover

US/Venezuela Workshop on HPC 2000



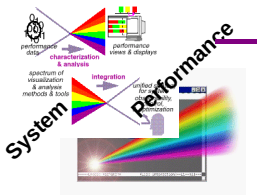
Acknowledgements

- NSF ACI-9624149
- DARPA DABT63-95-C-0072
- Co-PIs
  - Matt Mutka, Kurt Stirewalt
- Students
  - Aleks Bakic, Kuk-jin Lee, Abdul Waheed



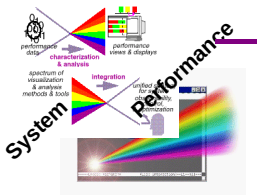
Outline

- What is performance visualization?
- Instrumentation systems
- Visualization systems
- PGRT infrastructure
  - Visual objects & VOML
- Uniform resource visualization
  - Components
  - Composition



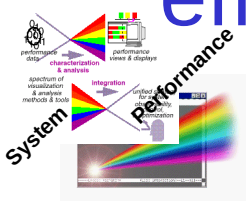
Outline

- What is performance visualization?
- Instrumentation systems
- Visualization systems
- PGRT infrastructure
  - Visual objects & VOML
- Uniform resource visualization
  - Components
  - Composition



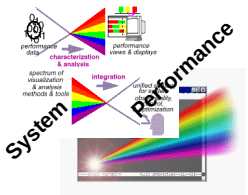
What is performance visualization?

- *Visualization*: the process of forming a mental picture or vision of something not present to the sight
- *Software visualization*: use of typography, graphic design, and animation, with modern human-computer interaction and computer graphics technology, to facilitate both the human understanding and effective use of computer software



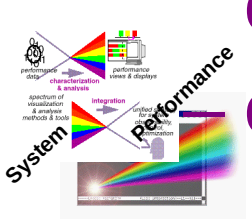
What is performance visualization?

- *Performance visualization*: a type of software visualization that includes aspects such as hardware performance (also *computation/system visualization*)
  - Visualization of concurrency is a specific concern for parallel and distributed systems
  - Used to evaluate performance, verify correctness, diagnose problems, and gain insight into structure and execution behavior



What is performance visualization?

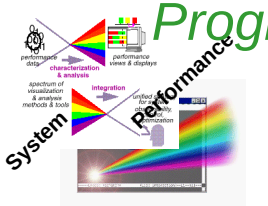
- Static representation of system state
  - structure/content of a large database
- Dynamic execution behavior
  - runtime control flow
  - memory usage
  - interprocessor communication
  - graphs of metrics, e.g., utilization, traffic
  - application-specific views of data and computations
  - etc.



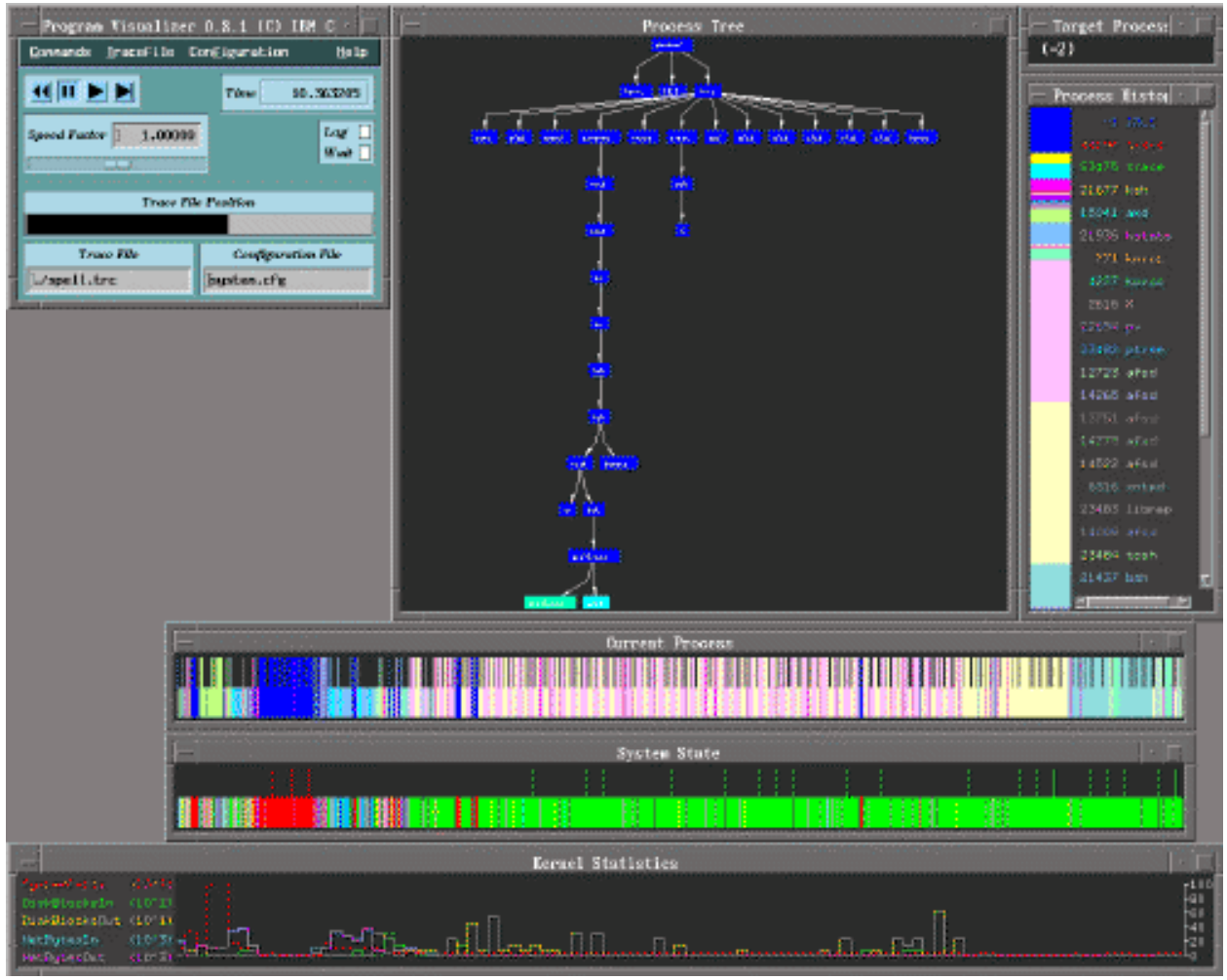
Examples

IBM's PV system configuration, showing activity on the system as a whole, including a process tree view, histogram of process runtimes, and strip charts of process and system activities.

Chapter 20 by Kimelman, Rosenburg, and Roth in Stasko, Domingue, Brown, and Price, *Software Visualization - Programming as a Multimedia Experience*, MIT Press, 1998.



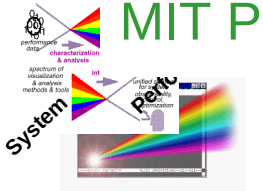
IBM's PV system configuration



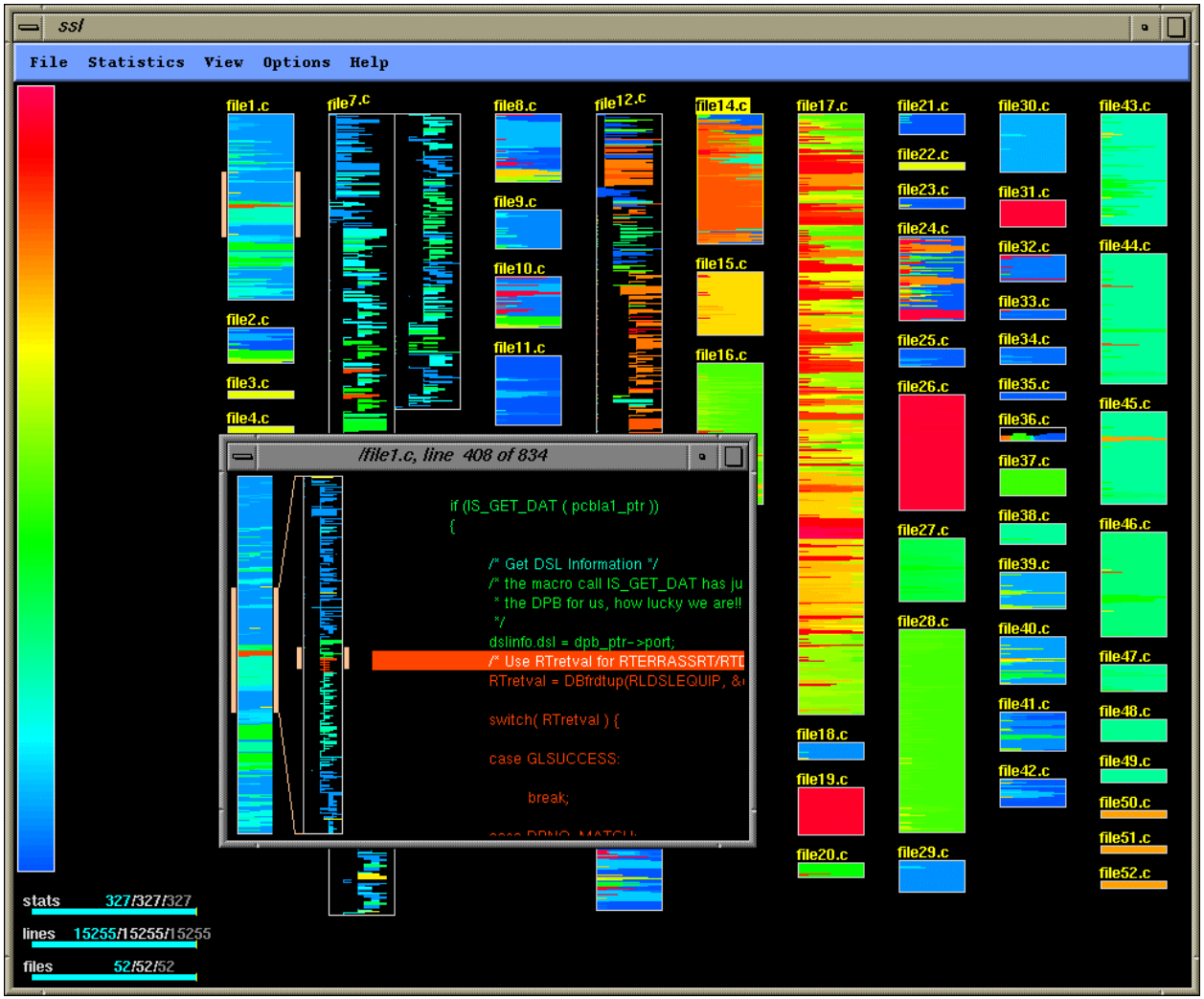
Examples

Lucent Technologies' SeeSoft code display, showing an overview of many files and their statistics at once, here code age. The newest code is in red and the oldest in blue. Blue and green files represent more stable code. SeeSoft visualizes text files by mapping each line into a thin row, colored according to a statistic of interest derived from version control systems, static analysis, and profiling.

Chapter 21 by Eick in Stasko, Domingue, Brown, and Price, *Software Visualization - Programming as a Multimedia Experience*, MIT Press, 1998.



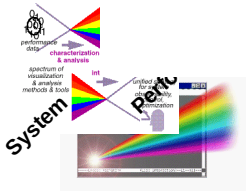
Lucent Technologies' SeeSoft code display



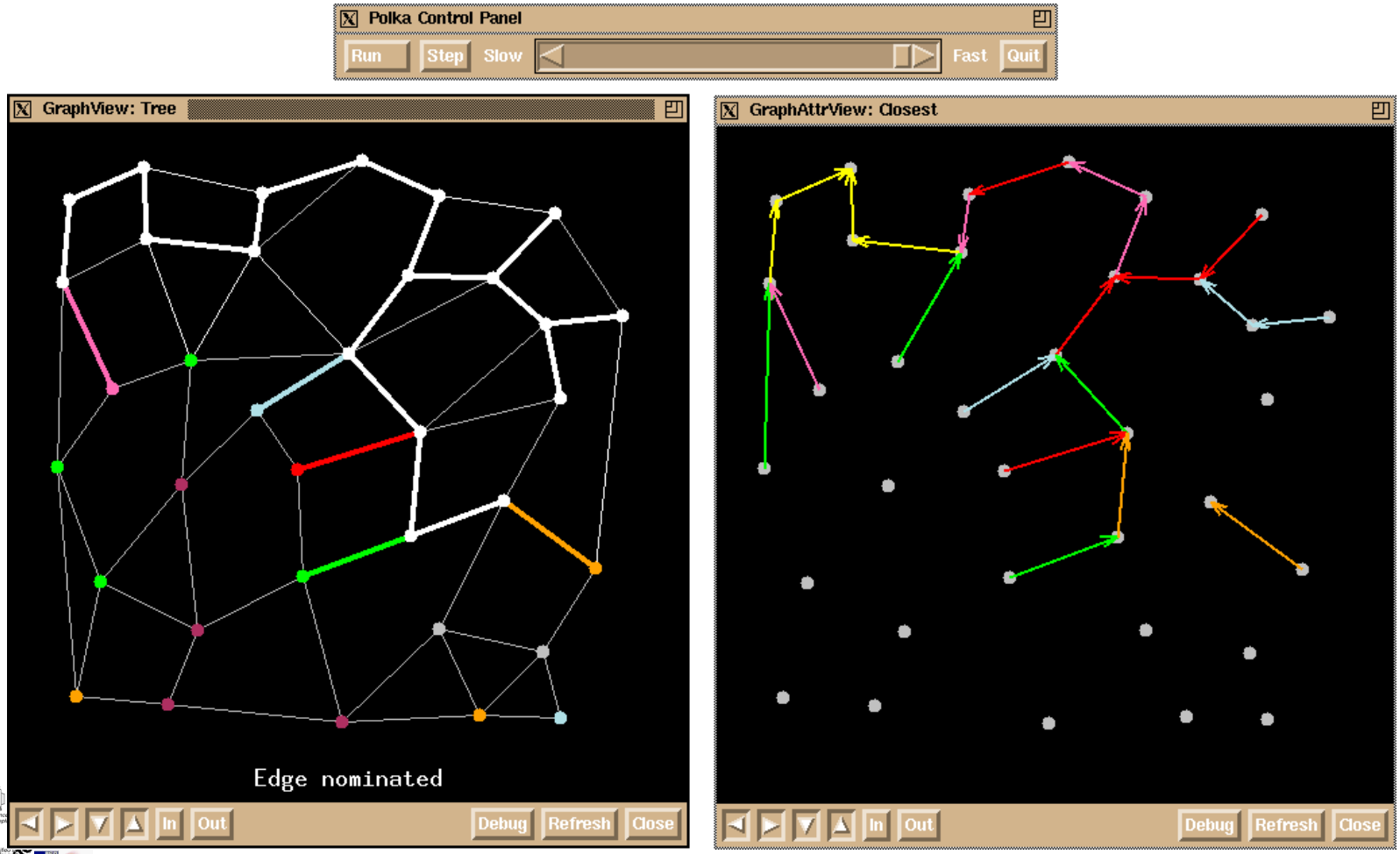
Examples

Georgia Institute of Technology's POLKA animation of a parallel minimum spanning tree program. The left view shows the graph and the spanning tree growing inside it. The right view shows the "closest" data structure maintained by the program.

Chapter 17 by Kraemer in Stasko, Domingue, Brown, and Price, *Software Visualization - Programming as a Multimedia Experience*, MIT Press, 1998.



Georgia Tech's POLKA animation

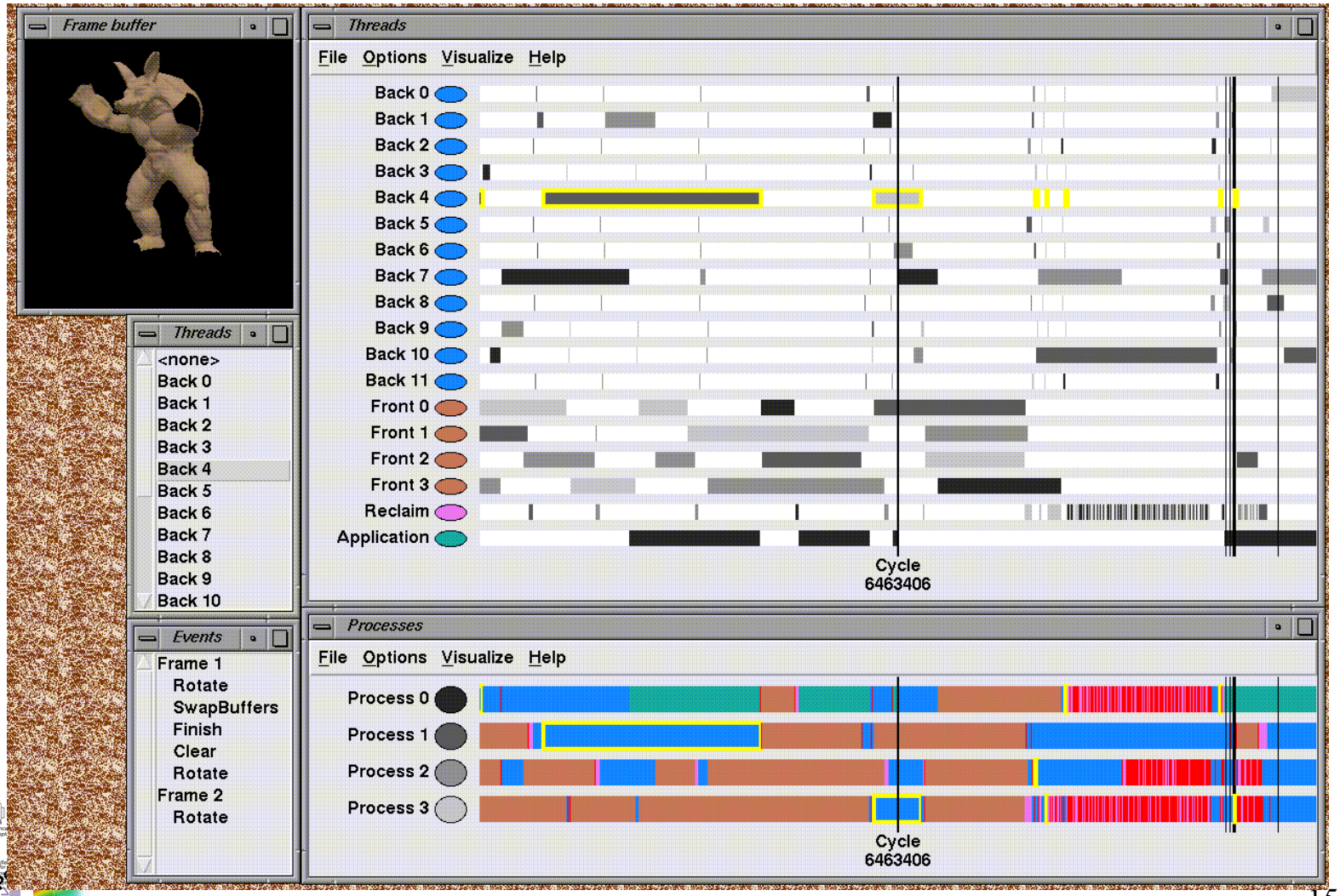


Examples

Stanford University's Rivet visualization of thread scheduling algorithms in the Argus parallel rendering library. This visualization combines Gantt charts of per-CPU and per-thread scheduling data, milestones in the rendering process, and a dynamic view of the framebuffer being generated by Argus. The dynamic display of the framebuffer synchronized to the data being displayed in the data windows provides a familiar application-based context for the user. The use of brushing between the various views allows otherwise disparate views to be used as a coherent whole.

Hanrahan, Rivet Project, Stanford University, http://www-graphics.stanford.edu/projects/rivet/.

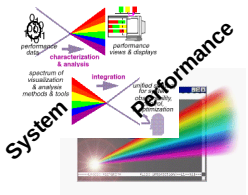
Stanford University's Rivet visualization



Think about …

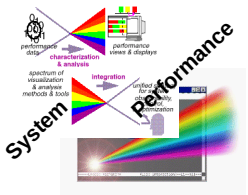
- How are these visualizations created?

- How are these visualizations used?



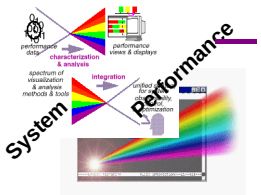
Steps in Visualization

1. Derive information from sources in the software development environment, static analysis of a program, and data collected during program execution
   - Instrumentation system collects data
2. Preprocess or analyze information
   - Format conversion, profiling, event-ordering, detection of abstract events and global states, etc.
3. Represent or display the program/system graphically
   - Visualization system supports creating and using views
   - Display issues: effectiveness, concurrency, scalabililty, application-specificity



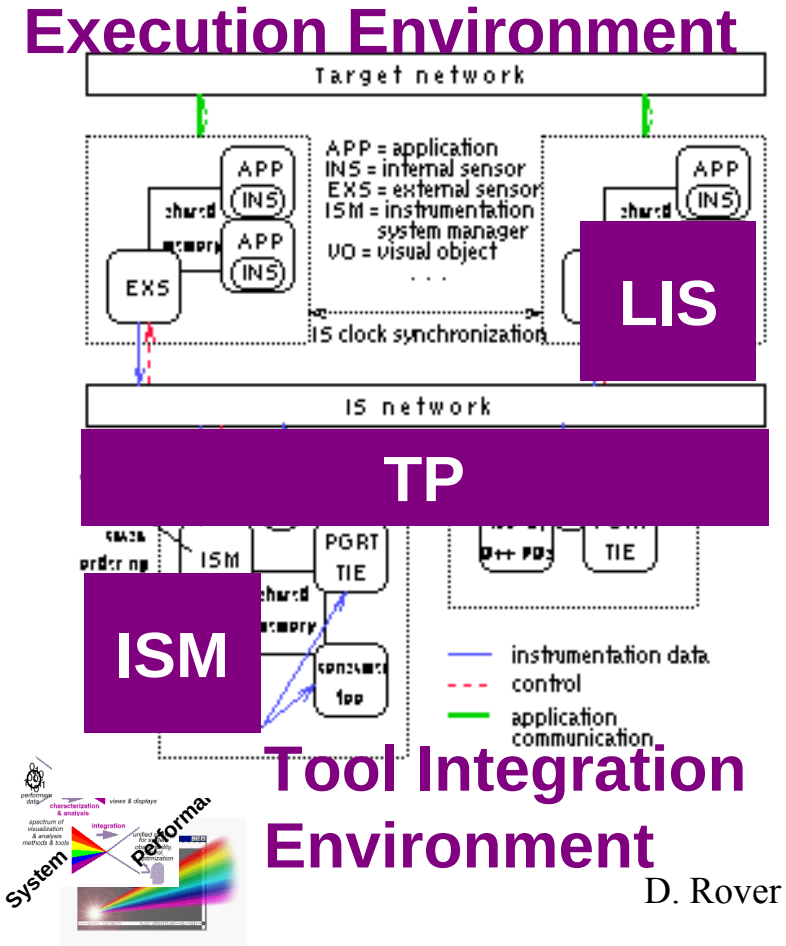
Outline

- What is performance visualization?
- Instrumentation systems
- Visualization systems
- PGRT infrastructure
  - Visual objects & VOML
- Uniform resource visualization
  - Components
  - Composition



Instrumentation Systems

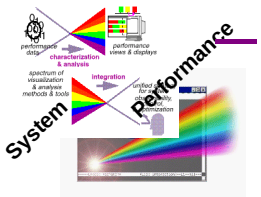
Baseline Reduced Instrumentation System Kernel (BRISK)



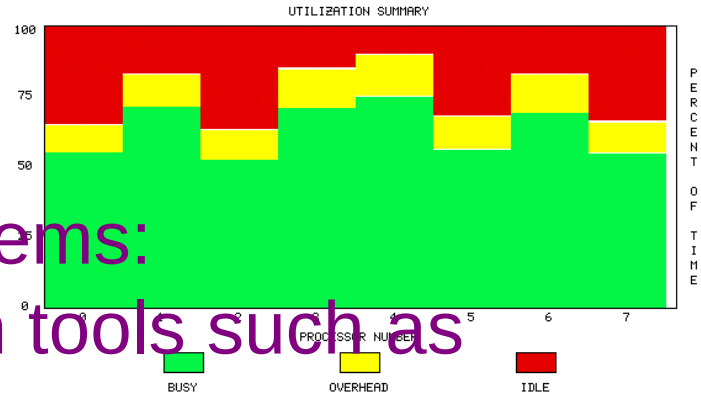
- Lightweight, distributed IS
- Portable, flexible LIS-ISM model
- Low-intrusion techniques
- LIS on a node: internal sensors in user processes, shared memory, external sensor
- ISM on same node as TIE, tools, CORBA extension

Outline

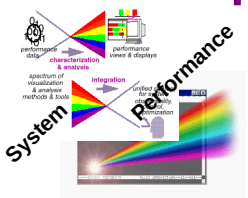
- What is performance visualization?
- Instrumentation systems
- Visualization systems
- PGRT infrastructure
  - Visual objects & VOML
- Uniform resource visualization
  - Components
  - Composition



Visualization Systems

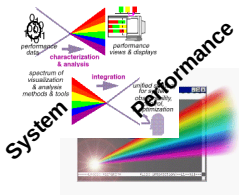


- History
  - Earliest visualization systems: performance visualization tools such as ParaGraph (late 1980's)
  - Application-specific visualizations of concurrent programs (early 1990's)
  - Integrated frameworks for diagnosing performance problems, such as TAU, Paradyn, and Pablo (mid 1990's)



Visualization Systems

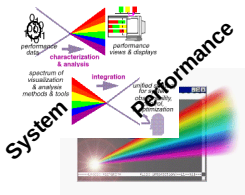
- Taxonomy
  - ***Scope*. What range of programs/machines can the system take as input for visualization?** For example, can the system visualize the concurrent features?
  - ***Content*, or *Abstraction*. What subset of information about the program/machine is visualized by the system?** For example, code versus control flow diagram.
  - ***Method of Specification*. How is the visualization specified?** Some systems provide fixed visualizations, while others let the user customize visualizations.



Chapter 1 by Price in Stasko, Domingue, Brown, and Price, *Software Visualization - Programming as a Multimedia Experience*, MIT Press, 1998.

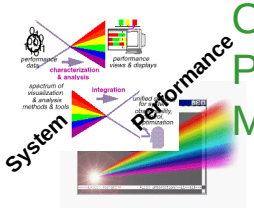
Visualization Systems

- Taxonomy (continued)
  - ***Interface*, or *Interaction*. How does the user of the system interact with and control it?** Navigation through a visualization is especially important for very large programs or data sets.
  - ***Presentation*, or *Form*. How is the system used to convey information?** For example, what medium and graphical elements are used in the visualization? Does the system provide facilities for eliding information? for synchronizing multiple views?



Visual Displays

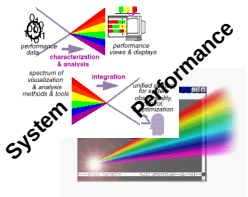
- Design Practices / Characteristics
  - ***Context*. Meaning of information to the user, relationships among information:** perspective, semantic context, subview mapping.
  - ***Scaling*. Size/dimensionality of graphical views as data sets become very large (e.g., as the number of processors or duration of execution becomes very large):** multi-dim. and multivariate representation, macroscopic and microscopic views, macro/micro composition and reading, adaptive graphical display, display manipulation, composite view.



Chapter 23 by Heath, Malony, and Rover in Stasko, Domingue, Brown, and Price, *Software Visualization - Programming as a Multimedia Experience*, MIT Press, 1998.

Visual Displays

- Design Practices (continued)
  - ***Perception and Interaction*. Interpretation of displays by user and support by tool for exploring execution behavior through views and options :** perception and cognition, observing patterns, user interaction.
  - ***Comparison*. Relationships between views or representations to gain insight into behavioral characteristics and their causes:** multiple views, small multiples, cross-execution views**.**

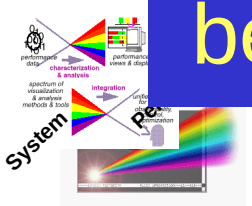


Visual Displays

- Design Practices (continued)
  - ***Extraction of Information*. Selection and presentation of useful information from large data sets:** reduction and filtering, clustering, encoding and abstracting, separating information.

- PV, Seesoft, POLKA, Rivet examples

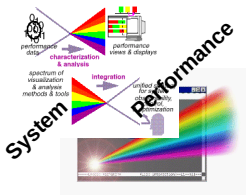
Visualizations and design practices can be classified.



Usability

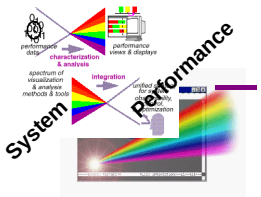
- Is a visualization system or graphical display effective?

- Definition
  - ISO definition: the effectiveness, efficiency, and satisfaction with which specified users can achieve specified goals in particular environments (ISO DIS 9241-11)
  - Software usability: … using particular computing systems



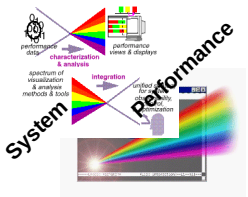
Usability

- *Developer push?* usability engineering, user-centric design
- Parallel Tools Consortium: researchers + developers + users
- *User pull?* Task Force on Requirements for HPC Software and Tools
  - Guidelines for including software and tools in HPC procurements (www.nacse.org/projects/HPCreqts/)
  - Performance tuning and debugging support



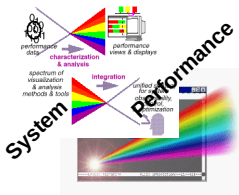
Reusability

- Definition
  - the extent to which a software component can be used (with or without adaptation) in multiple problem solutions
  - Reuse:
    - synthesizing a solution to a problem based on predefined solutions to subproblems
    - implementing new software systems from pre-existing software, e.g., an archive of software artifacts
  - Why?
    - Reduce code development effort (increase productivity)
    - Improve quality of the software



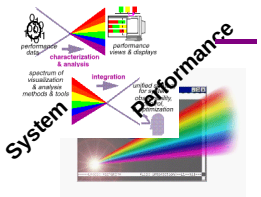
Reusability

- Issues
  - Design for reuse is expensive.
  - Tools that originate in university research labs are less likely to have reuse as a goal.
    - Object-oriented and component-based approaches, such as Java or DCOM, may impact reusability favorably.
  - Some tools, such as Pablo, have re-usable modules.
    - This type of reuse may conflict with usability for the typical user (i.e., a tool that is highly configurable may appear complex to a user).
    - Other types of reuse, such as look/feel of a GUI, tend to affect usability positively.
  - Development of tools using a design-for/with-reuse strategy is a long-term outlook.
    - Interoperable tools
    - Extensible environments

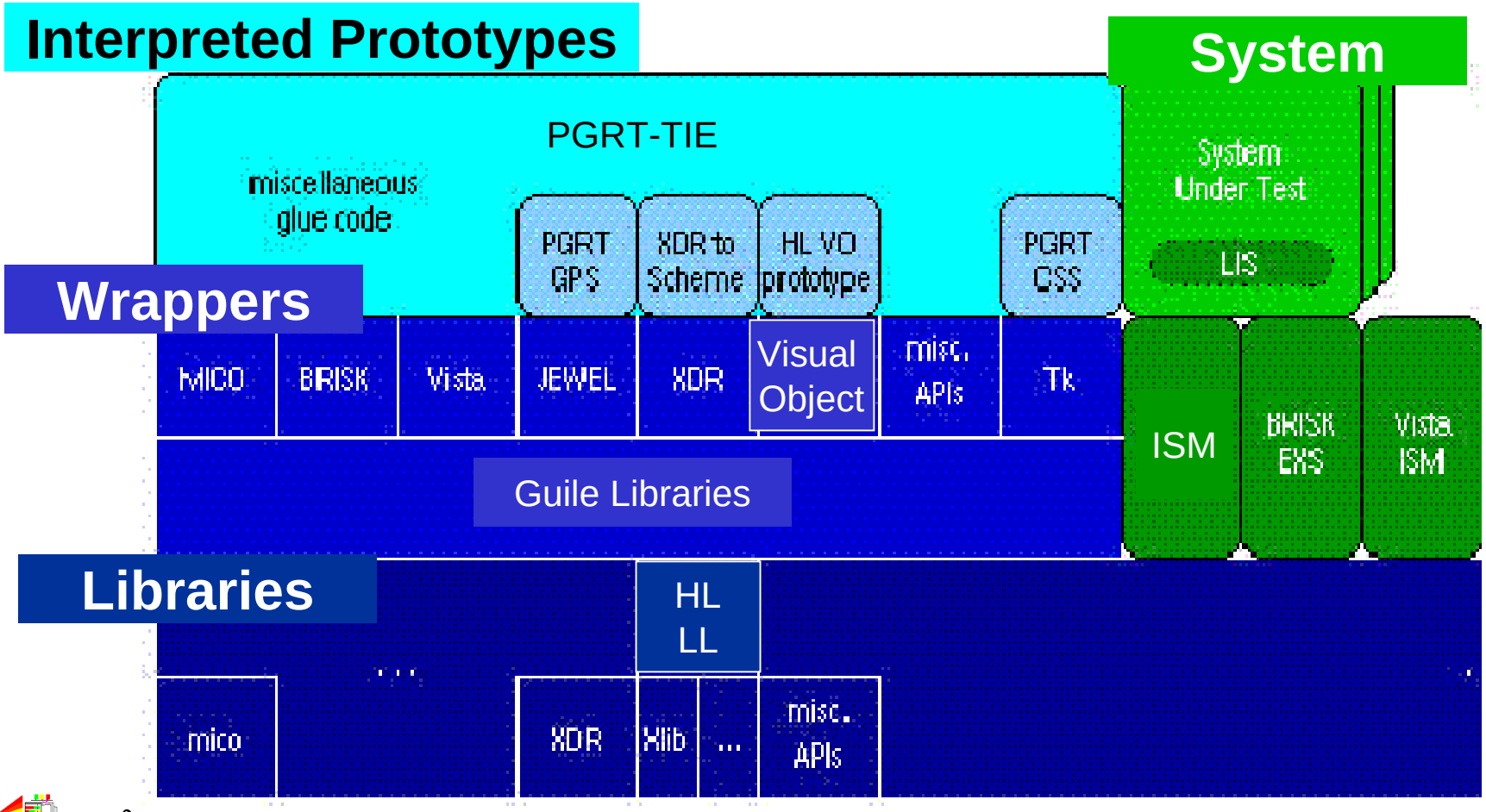


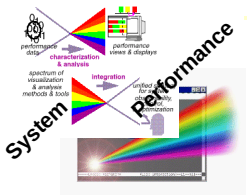
Outline

- What is performance visualization?
- Instrumentation systems
- Visualization systems
- PGRT infrastructure
  - Visual objects & VOML
- Uniform resource visualization
  - Components
  - Composition



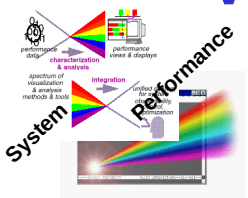
PGRT Infrastructure



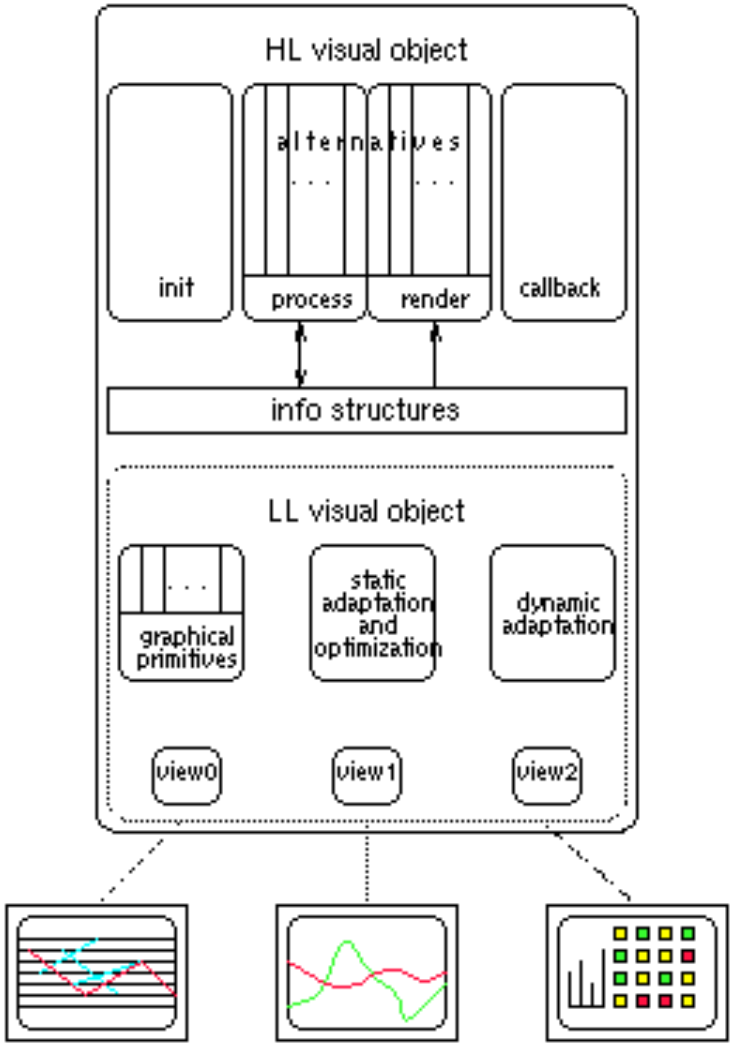


TIE: Tool Integration Environment

- Integrates software components using a language-based glue-and-library model
- Based on Guile software environment
  - Scheme-based system
  - Extensive library, interfaces and features
  - Multilevel integration options
- Interfaces to instrumentation systems, visualization tools, XDR specifications

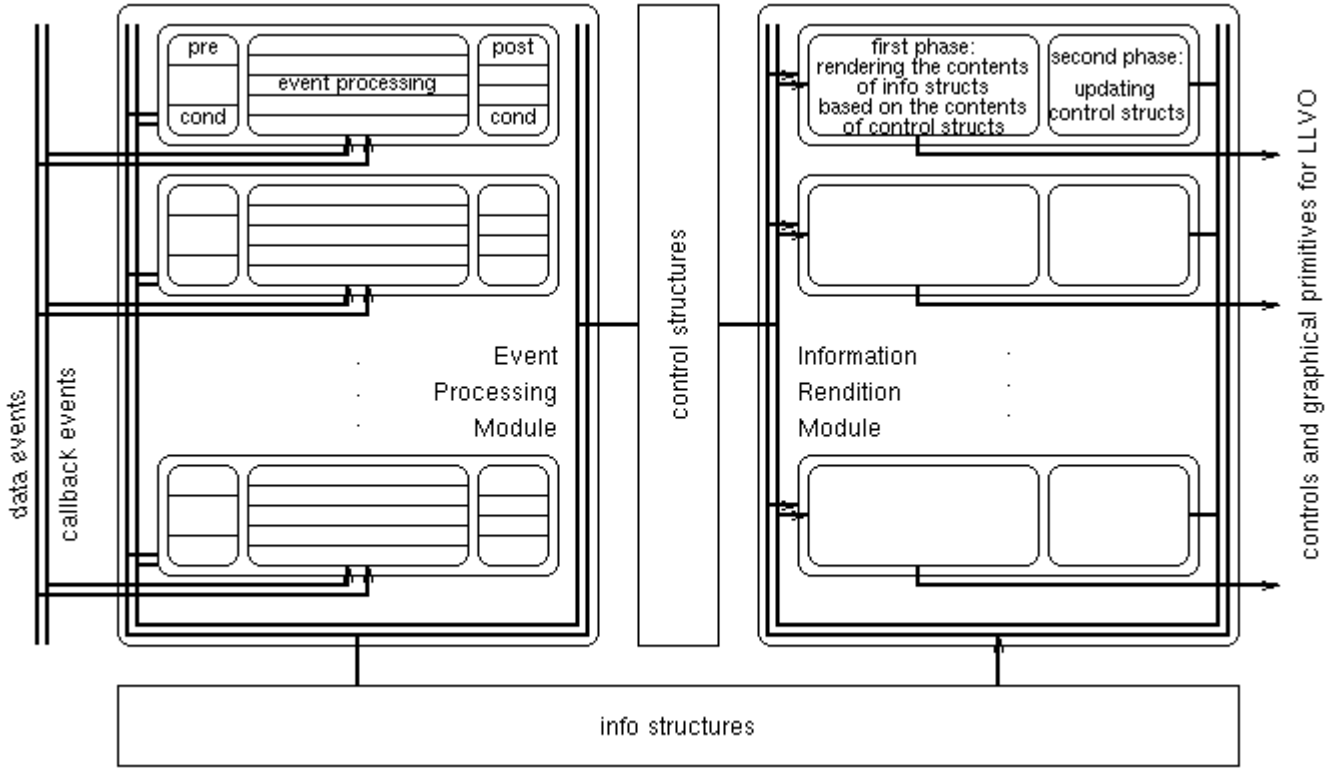


Visual Objects



EPIRA

Event Processing & Information Rendering Architecture for High-Level Visual Objects (HLVOs)

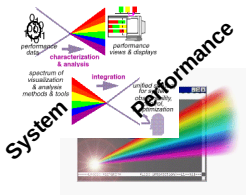


VOML

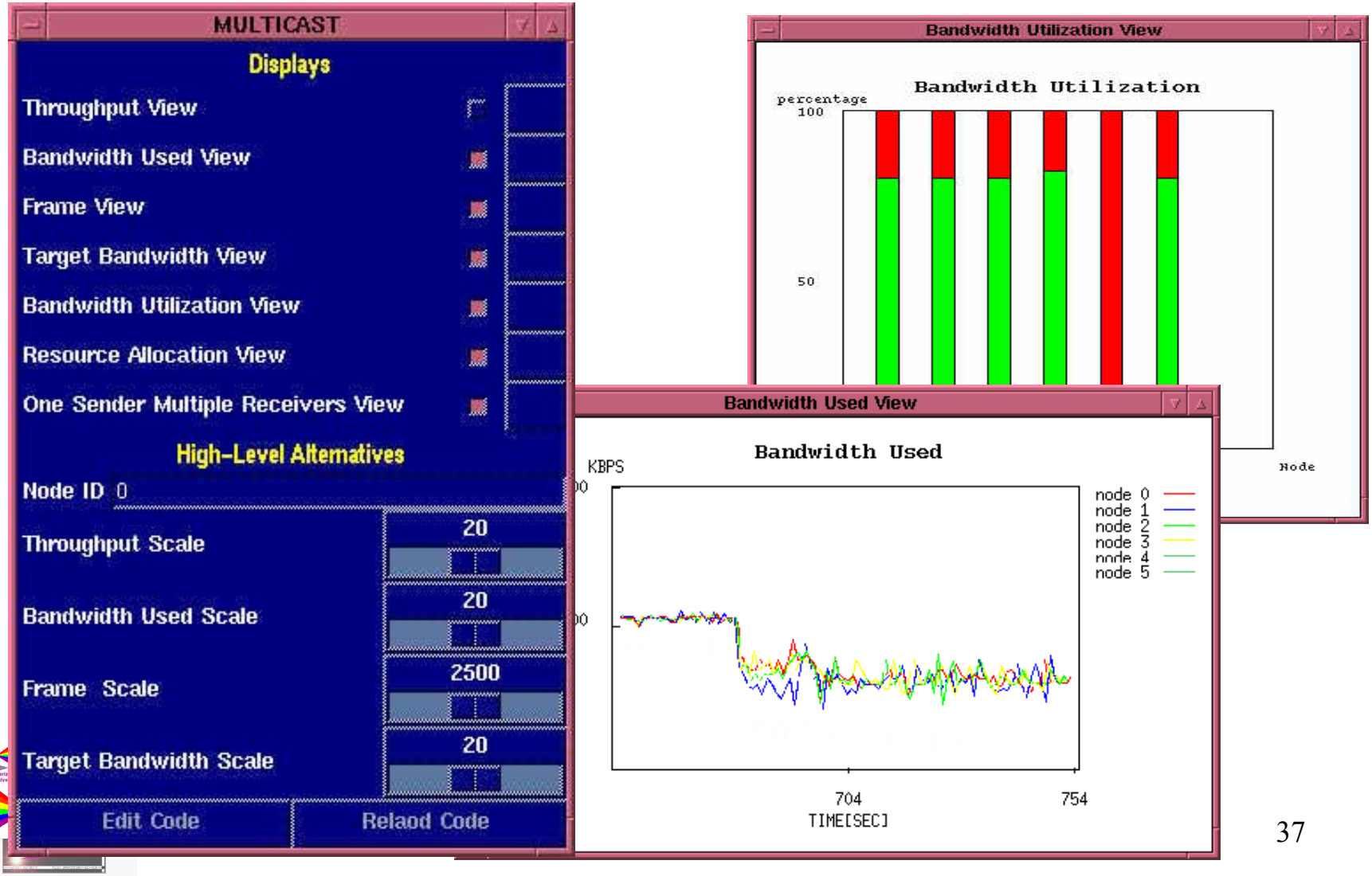
Visual Object Markup Language

- **Prototyping of HLVOs**
- **SGML Document Type Definition**
  - **Higher level elements**
  - **Relations between elements**
- **Reusable software components**
- **Component semantics via embedded Scheme code**

```
voml
  head
  body
    visual-object
      event-declarations
        data-event
      info-structures
      control-structures
      utility-code
      view-initializations
        view
      event-processing
        ep-component
          preprocess-inputs
      info-rendition
        ir-component
```

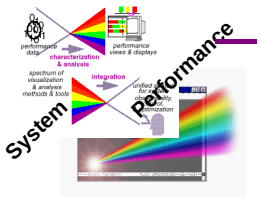


PGRT Displays



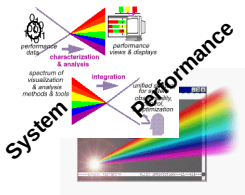
Outline

- What is performance visualization?
- Instrumentation systems
- Visualization systems
- PGRT infrastructure
  - Visual objects & VOML
- Uniform resource visualization
  - Components
  - Composition



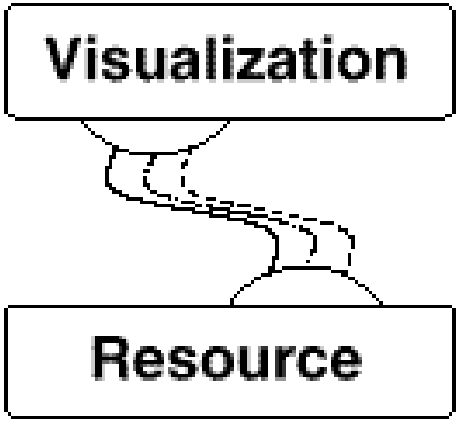
Uniform Resource Visualization

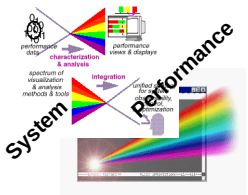
- URV is a component-based strategy
  - Constructing performance visualizations
    - Composing system-level views
    - Sharing visualization design knowledge
    - Describing visualizations and their interfaces uniformly
  - Monitoring and analyzing parallel & distributed systems
    - Viewing different resources, levels in a coordinated framework



Visualization Construction

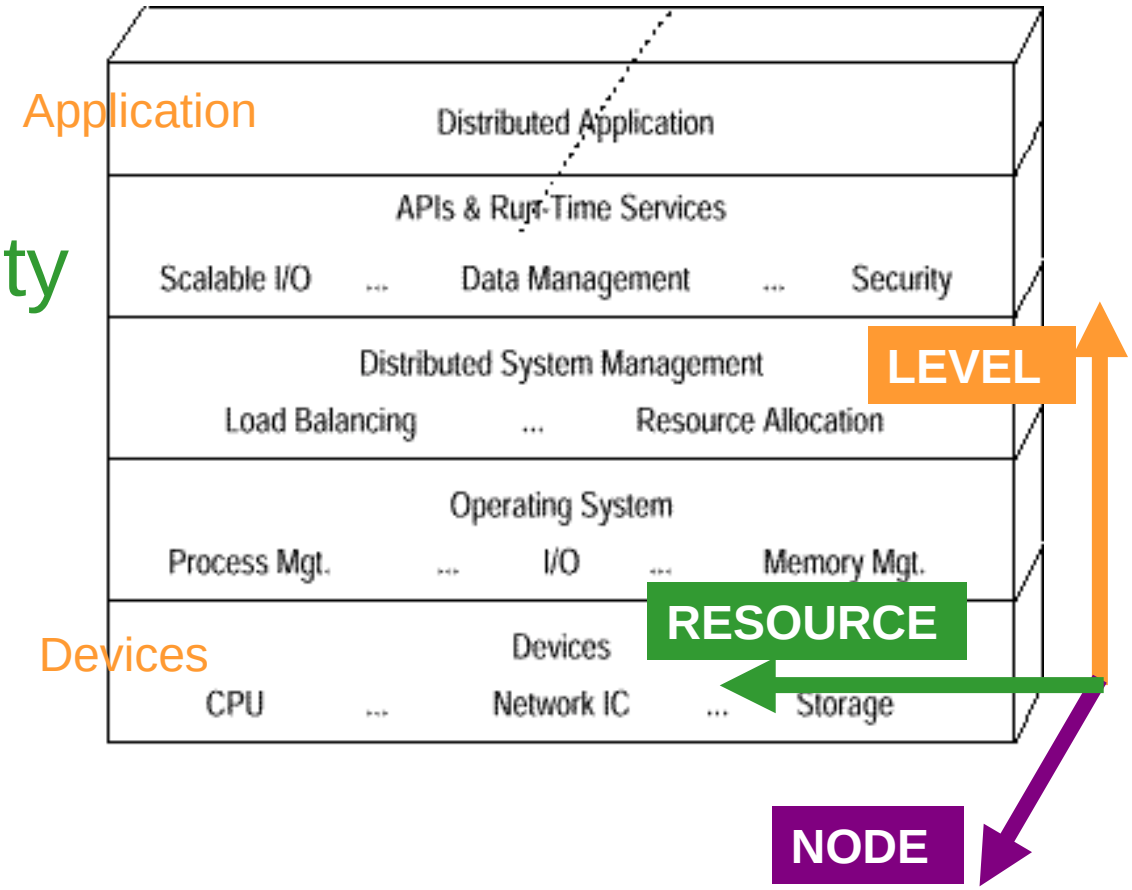
- Two independently executing components
  - Resource-monitoring component
    - Physical entity, e.g., processor
    - Logical entity, e.g., array
  - Visualization component
  - Connected by a collaboration
  - Mapped onto EPIRA
  - Written within VOML



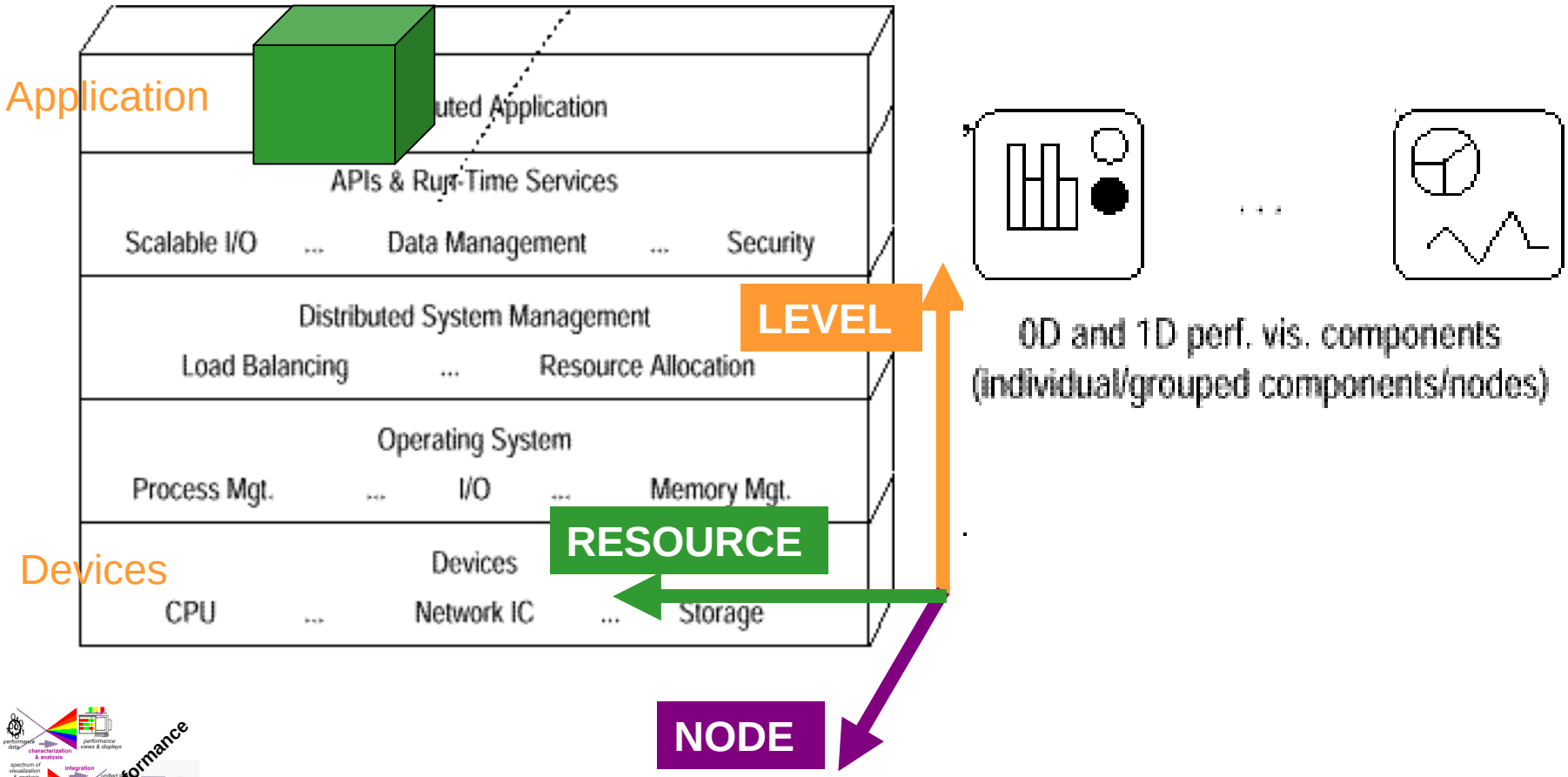


System Visualization

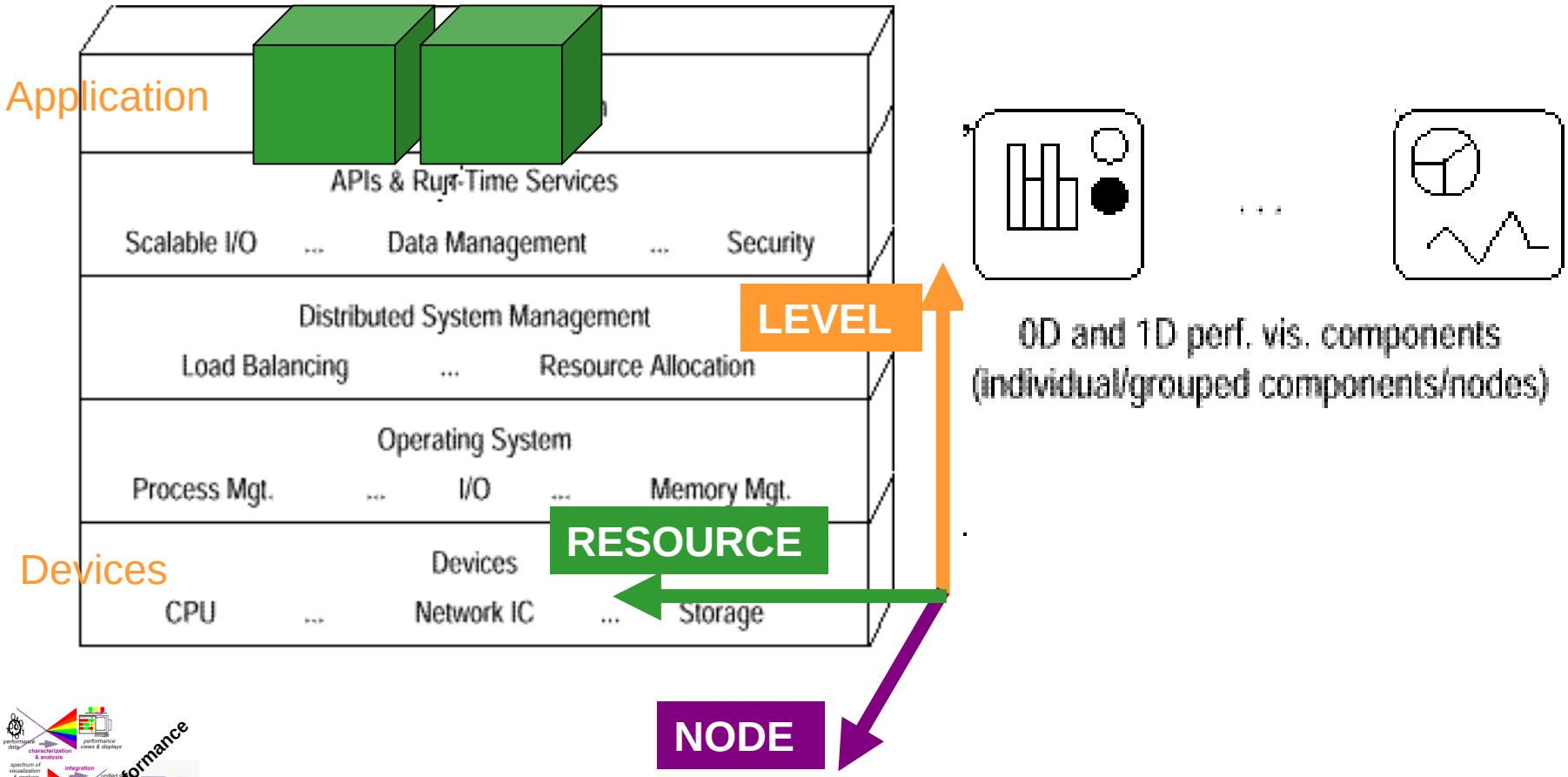
- Scale
- Hierarchy
- Heterogeneity
- Dynamics



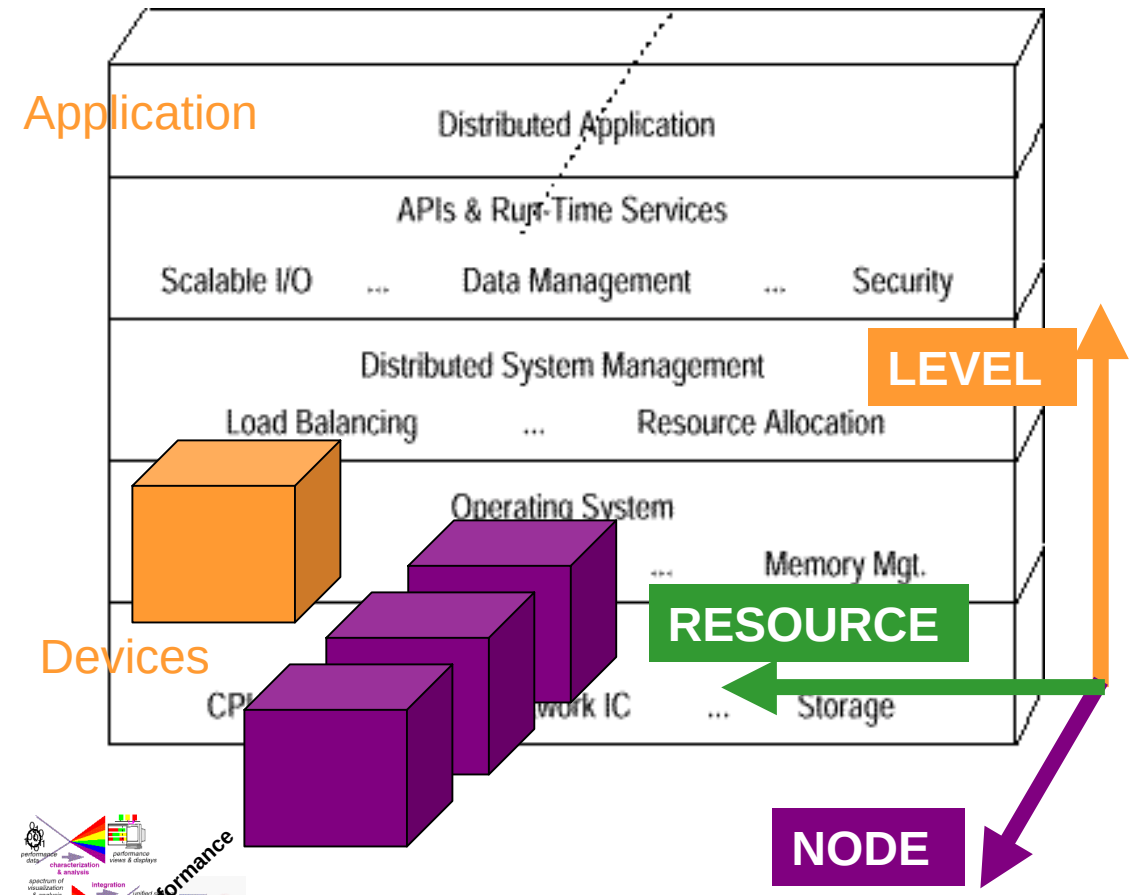
URV

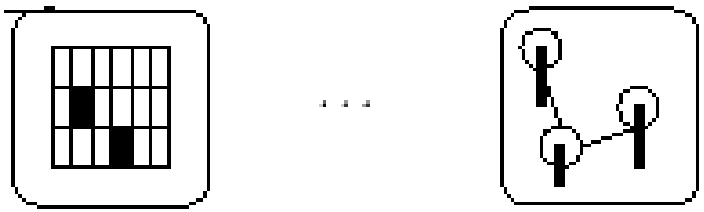

Application
uted Application
APIs & Run-Time Services
Scalable I/O ... Data Management ... Security
Distributed System Management
Load Balancing ... Resource Allocation
Operating System
Process Mgt. ... I/O ... Memory Mgt.
Devices
CPU ... Network IC ... Storage
LEVEL
RESOURCE
NODE
0D and 1D perf. vis. components
(individual/grouped components/nodes)

URV

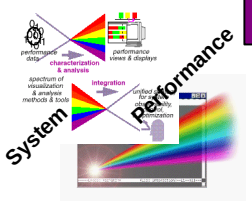

Application
APIs & Run-Time Services
Scalable I/O ... Data Management ... Security
Distributed System Management
Load Balancing ... Resource Allocation
Operating System
Process Mgt. ... I/O ... Memory Mgt.
Devices
CPU ... Network IC ... Storage
LEVEL
RESOURCE
NODE
0D and 1D perf. vis. components
(individual/grouped components/nodes)

URV

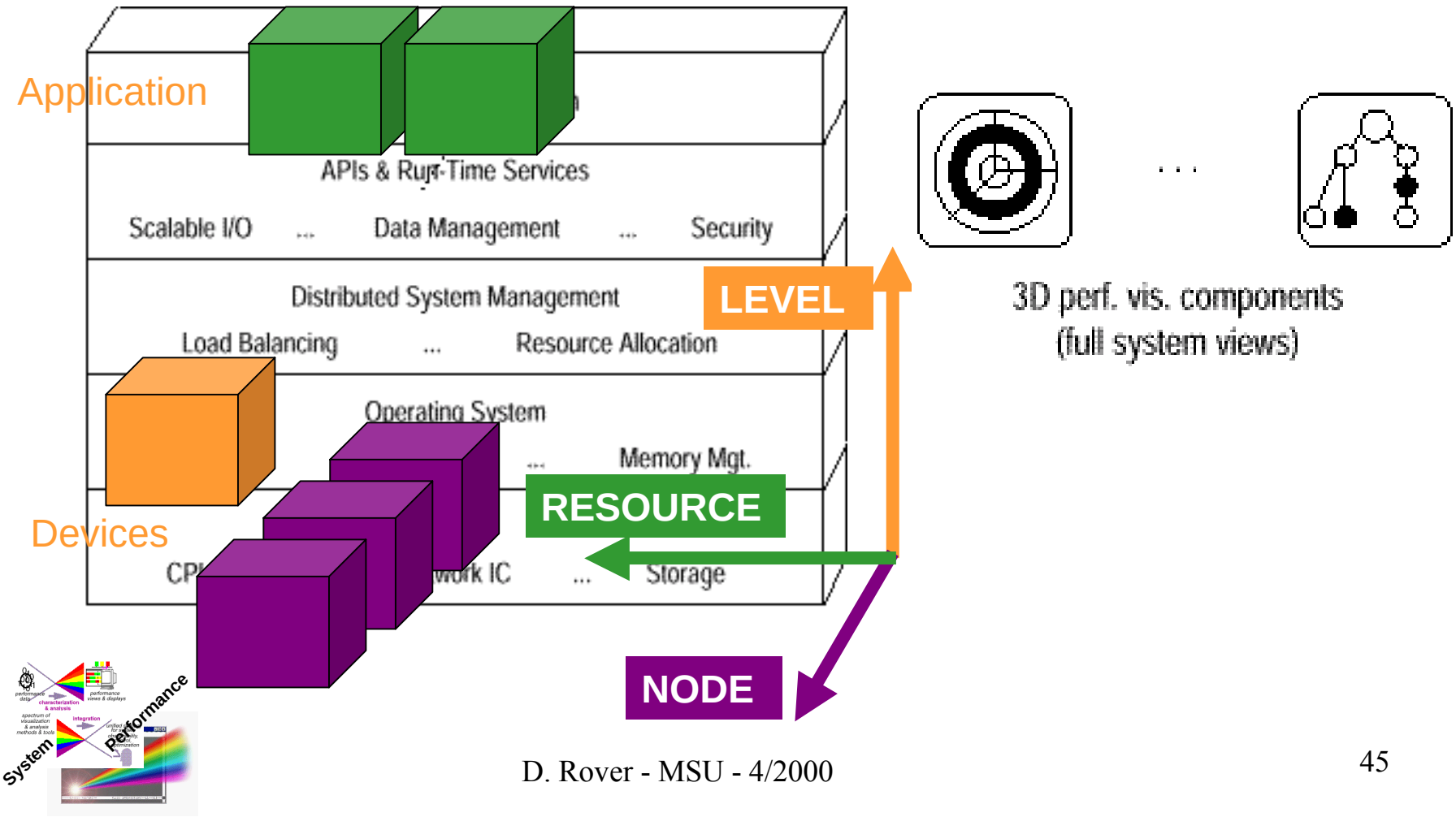




2D perf. vis. components
(layers/subsystems)

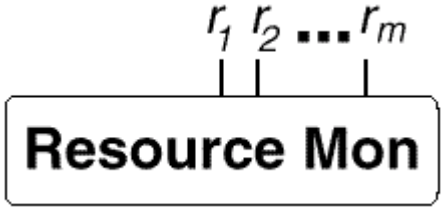


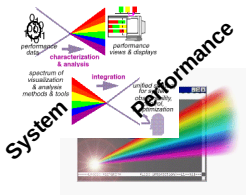
URV


Application
APIs & Run-Time Services
Scalable I/O ... Data Management ... Security
Distributed System Management
Load Balancing ... Resource Allocation
Operating System
... Memory Mgt.
Devices
CPU
Network IC ... Storage
LEVEL
RESOURCE
NODE
3D perf. vis. components
(full system views)
System
Performance

Visualization Standards

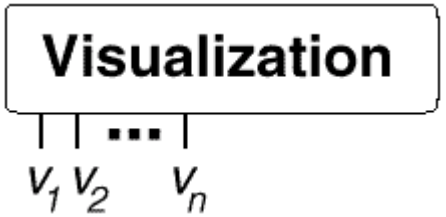
- Resource Mon component
  - Monitoring and control services for a resource

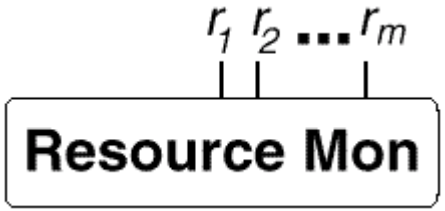


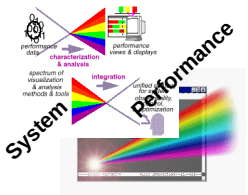


Visualization Standards

- Resource Mon component
  - Monitoring and control services for a resource
- Visualization component
  - Implementation of VOML high-level visual object

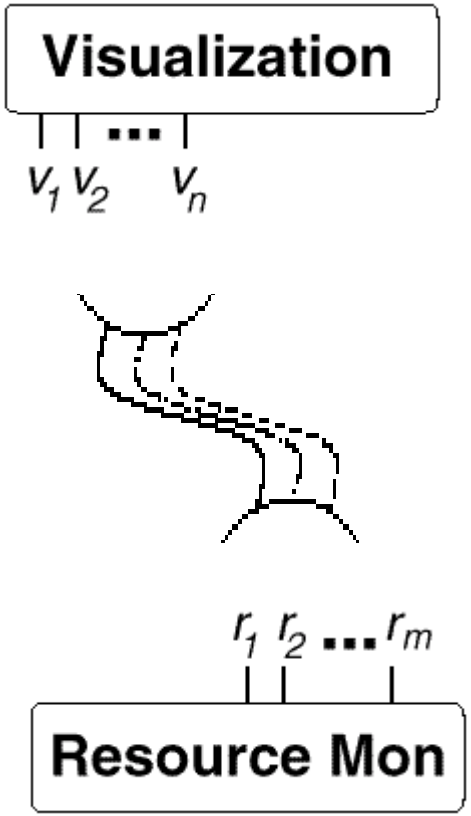


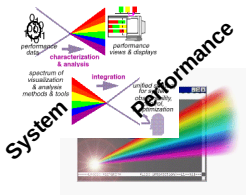




Visualization Standards

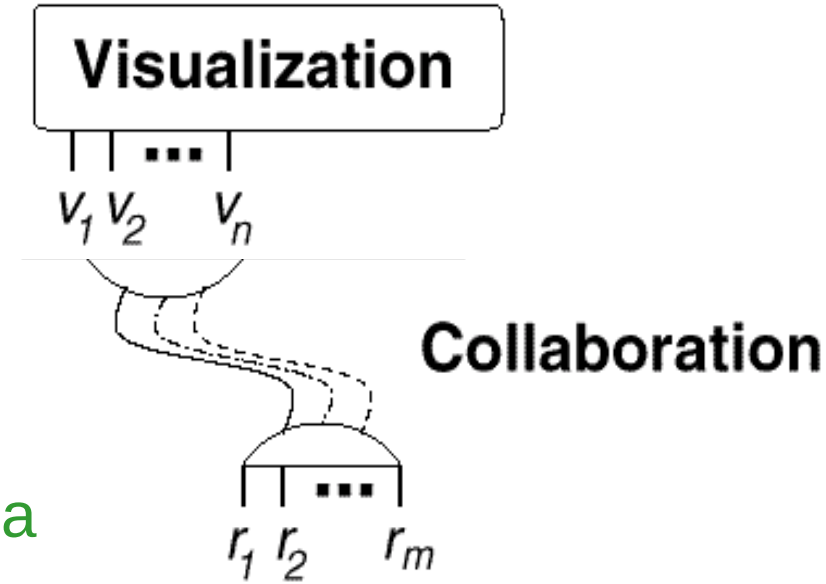
- Resource Mon component
  - Monitoring and control services for a resource
- Visualization component
  - Implementation of VOML high-level visual object
- Component interface
  - Set of services provided to and required of other components

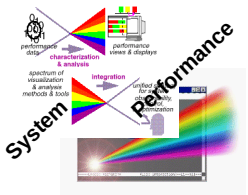




Plug-in Metaphor

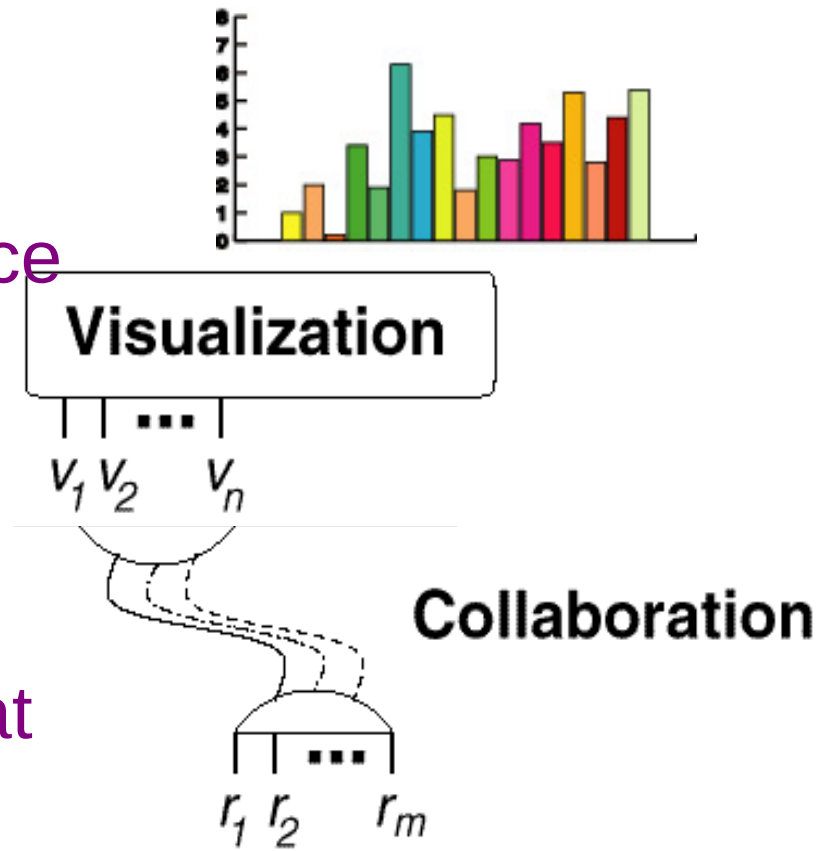
- Collaboration
  - Description of connection between services
  - More complex than binding the names of services
    - Buffer types
    - Synchronization between a single resource and multiple visualizations

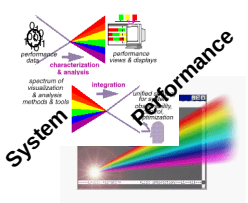




Plug-in Metaphor

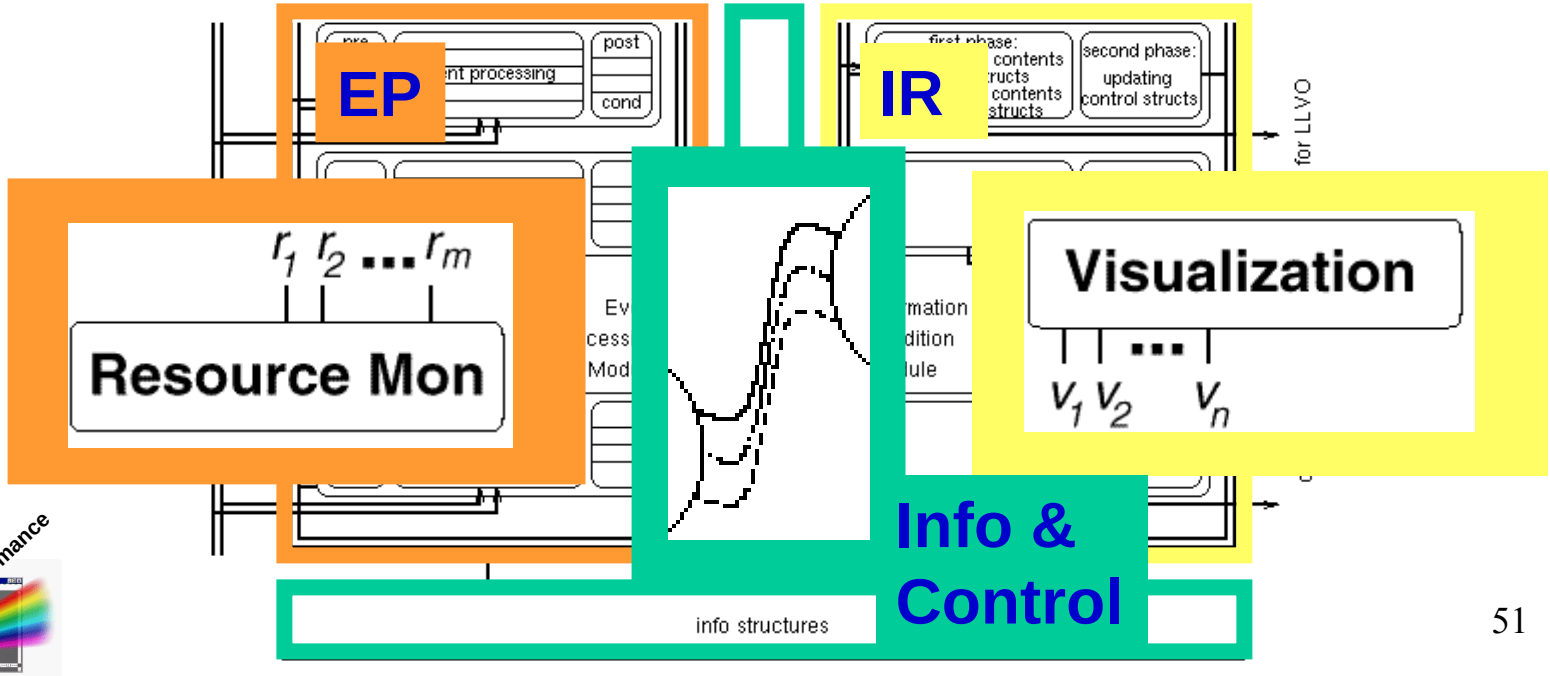
- Standardization
  - Specialize a generic visualization to an interface via a collaboration
    - Services $\{v_1, v_2, \ldots, v_n\}$ provide generic data flows
  - Reuse visualization with any resource in family that shares same interface
    - Resource interface defines services $\{r_1, r_2, \ldots, r_n\}$

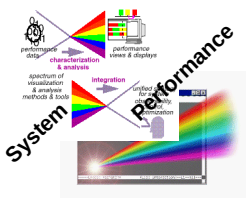




EPIRA Mapping

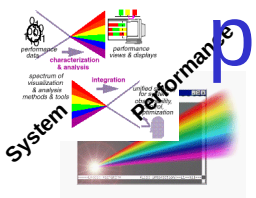
- Visualization components --> IR
- Resource-mon components --> EP
- Collaborations --> Info & Control Structures





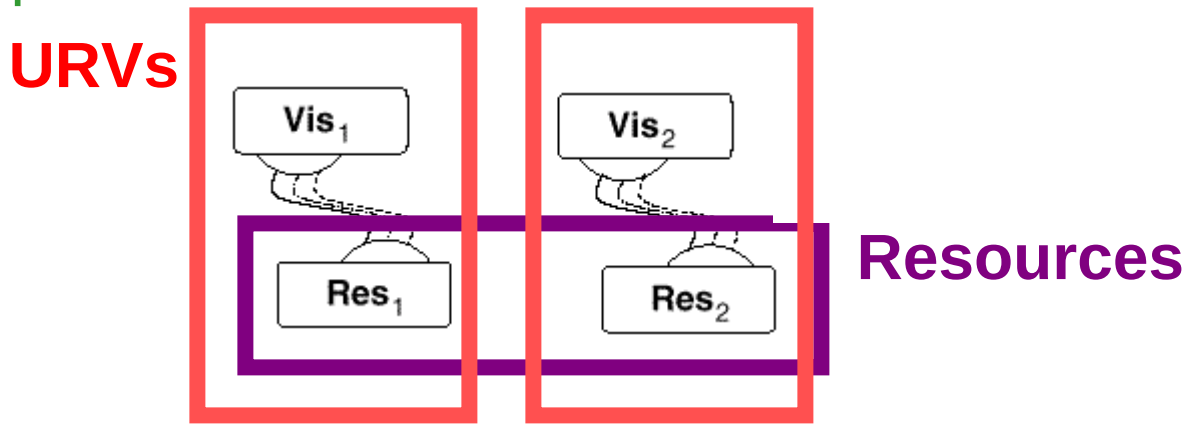
Visualization Design Knowledge

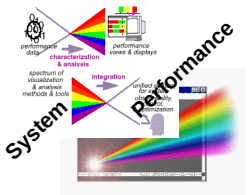
- Description of a component
  - Concept: what the component does
  - Content: how the concept is implemented and can be specialized
  - Context: the domain in which the component may be applied
- Searchable component specifications
- Reusable design features, visualization principles



Visualization Composition

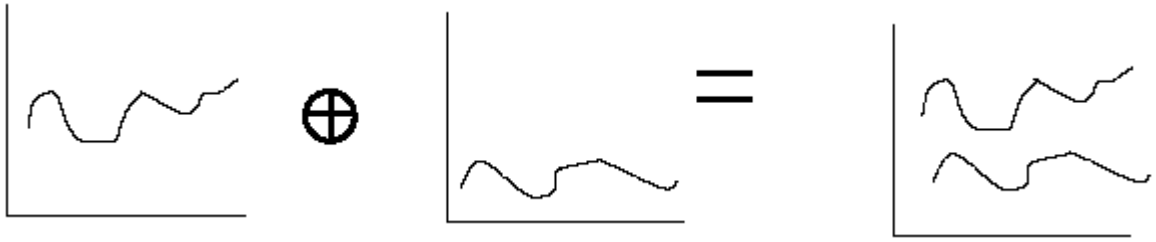
- Composition creates system-level views representing multiple resources.
  - Two resources, Res_1 and Res_2
  - Visualization issues
    - Synchronization
    - Abstraction

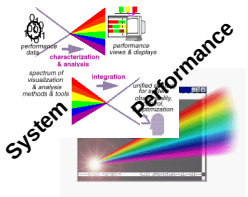




Visualization Composition

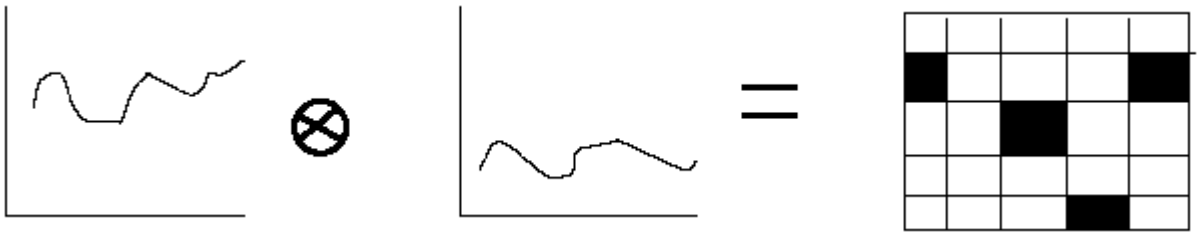
- Synchronization
  - Separate views may cause incorrect correlations
    - Interprocess communication
    - Network traffic
  - Composition: synchronize/join (spatially and temporally) into a single graph based on a common independent variable

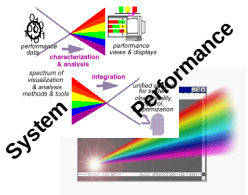




Visualization Composition

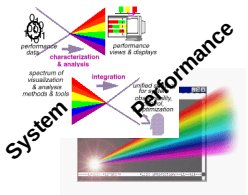
- Abstraction
  - Separate views may hide multidimensional information and relationships
    - Interprocess communication
    - Network traffic
  - Composition: synthesize information
    - Process matrix of network traffic





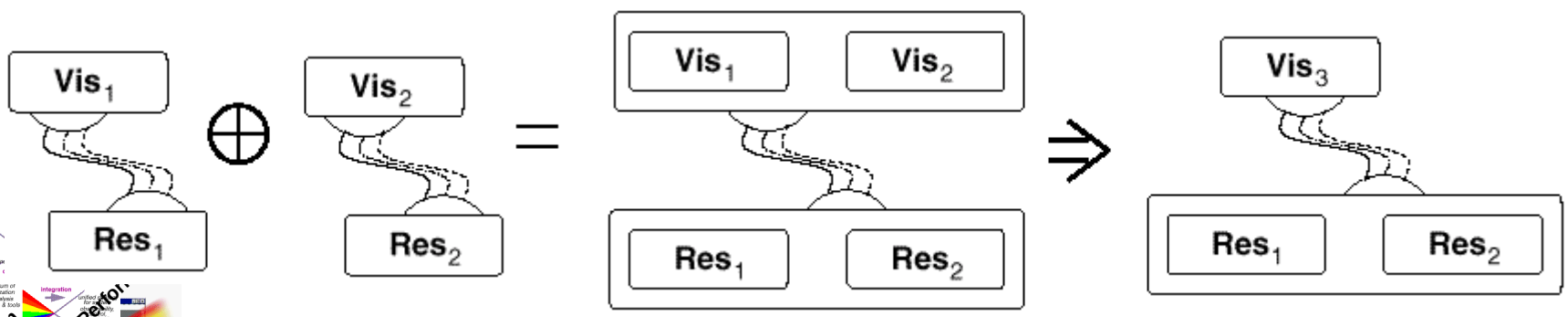
Visualization Composition

- Two types
  - Union ⊕: set of concrete transformation rules
    - Transformational approach: binary function takes two URVs and constructs a third URV
    - Composition transform: matches multiple URV view descriptions and replaces them with a synchronized, but functionally equivalent, view
  - Synthesis ⊗: design activity supported by a framework



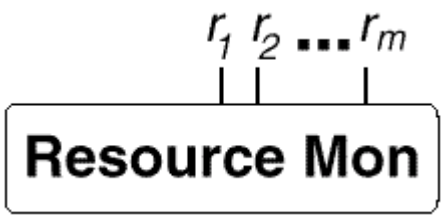
Union ⊕

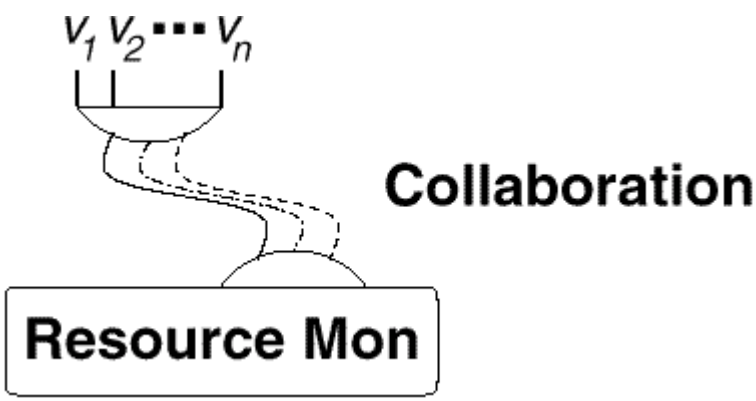
- Aggregate of Res_1 and Res_2 unchanged
- Collaboration replaced via transformation
- Visualization replaced via transformation
  - Aggregate of separate visualization components Vis_1 and Vis_2
  - Replaced by new visualization Vis_3



Union ⊕

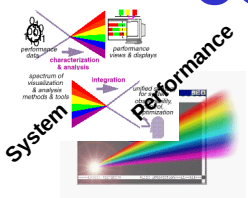
- Services $\{r_1, r_2, \ldots, r_n\}$ define the interface to the aggregate resource
- Aggregate resource is coupled with a collaboration that names services $\{v_1, v_2, \ldots, v_n\}$
- Interface to new visualization component is plug-compatible with $\{v_1, v_2, \ldots, v_n\}$



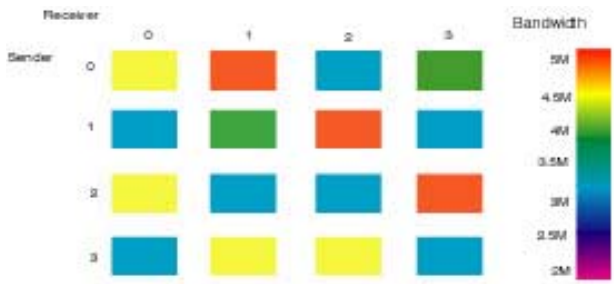


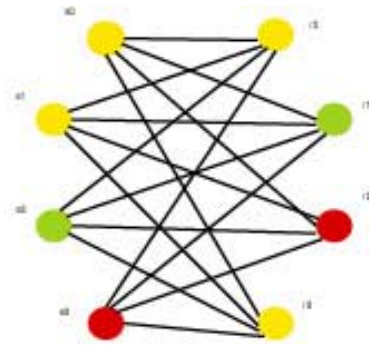
Synthesis ⊗

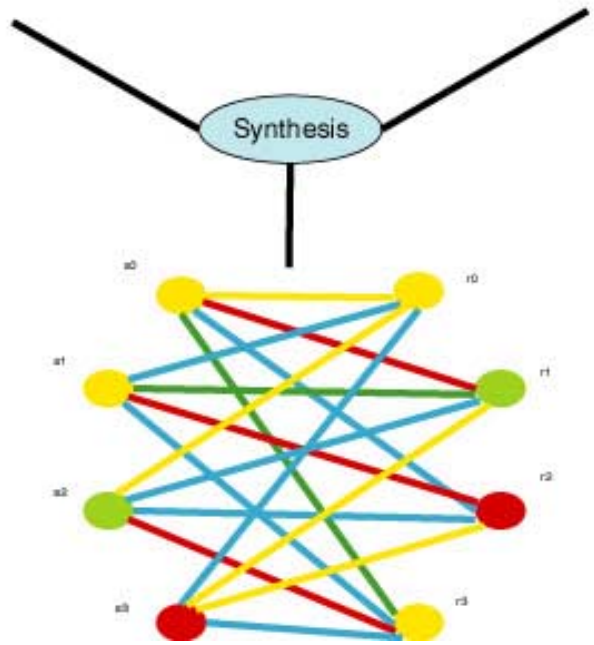
- Conscious activity by a human designer that cannot be automated fully
- Indexing of URVs and visualization components according to a classification scheme
- Designer specifies a set of search attributes to guide the search for a component

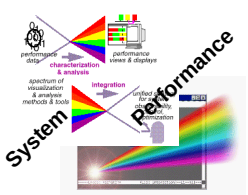


Synthesis ⊗

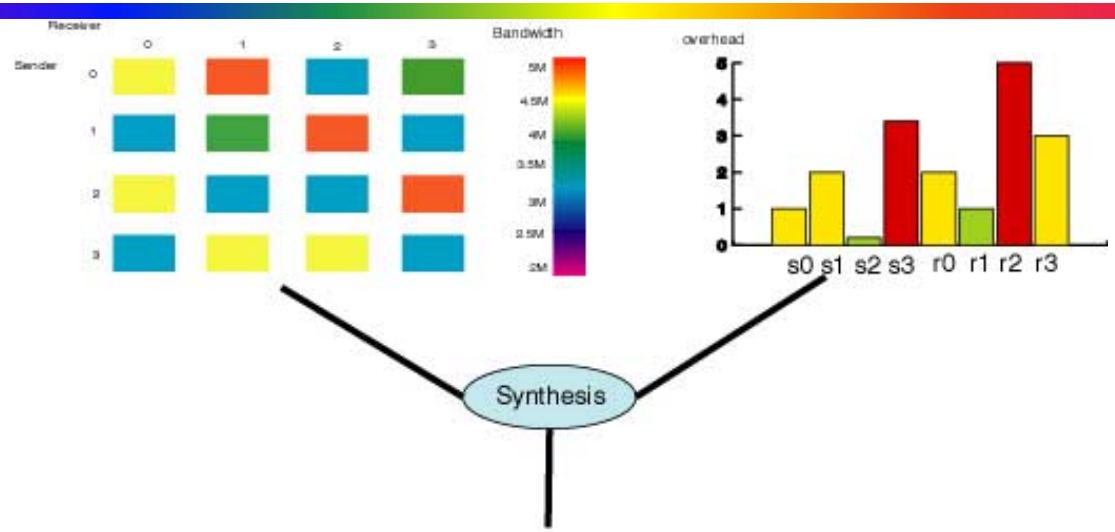

Receiver
0
1
2
3
Sender
0
1
2
3
Bandwidth
5M
4.5M
4M
3.5M
3M
2.5M
2M

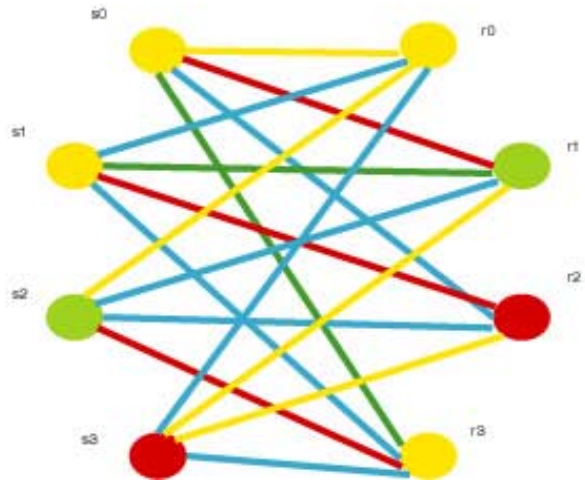



Synthesis
s0
s1
s2
s3
r0
r1
r2
r3


System
Performance

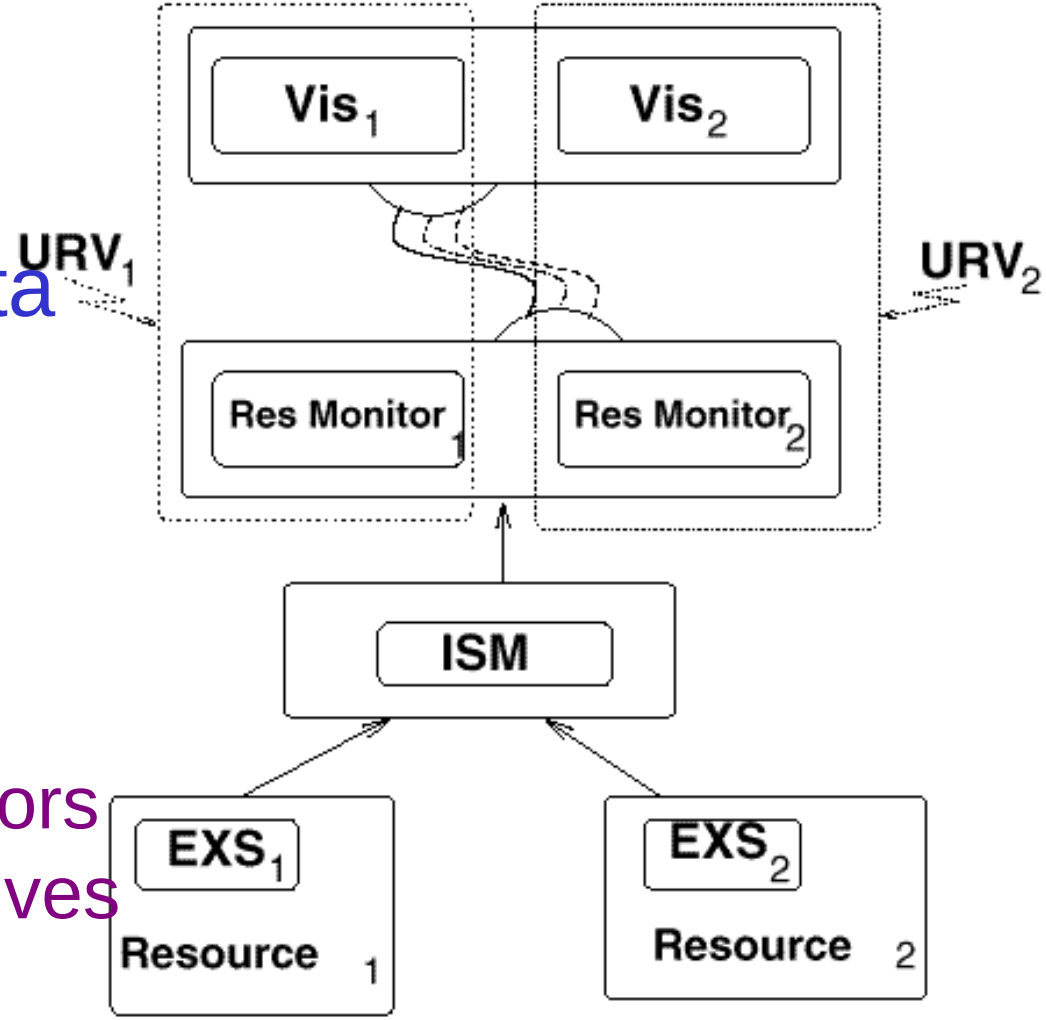
Synthesis ⊗


Receiver
Sender
0
1
2
3
Bandwidth
5M
4.5M
4M
3.5M
3M
2.5M
2M
overhead
6
4
3
2
1
0
s0 s1 s2 s3 r0 r1 r2 r3
Synthesis


s0
s1
s2
s3
r0
r1
r2
r3

Instrumentation System Support

- Challenge: integration of performance data from unrelated resources
  - ISM: proxy for resources
  - Resource monitors register themselves with the ISM



Features of URV

- Usability ☺ Reusability ❒
- ❒ **Multi-level, composable, reusable, distributable components (URV software architecture)**
- ❒ ☺ **Uniform descriptions of views for hardware/software resources**
- ❒ ☺ **Framework for developers to supply and catalog URVs**
- ❒ ☺ **Rule-based visualization composition**
- ☺ **Access to performance views**
- ❒ ☺ **Capture of visualization design practices**