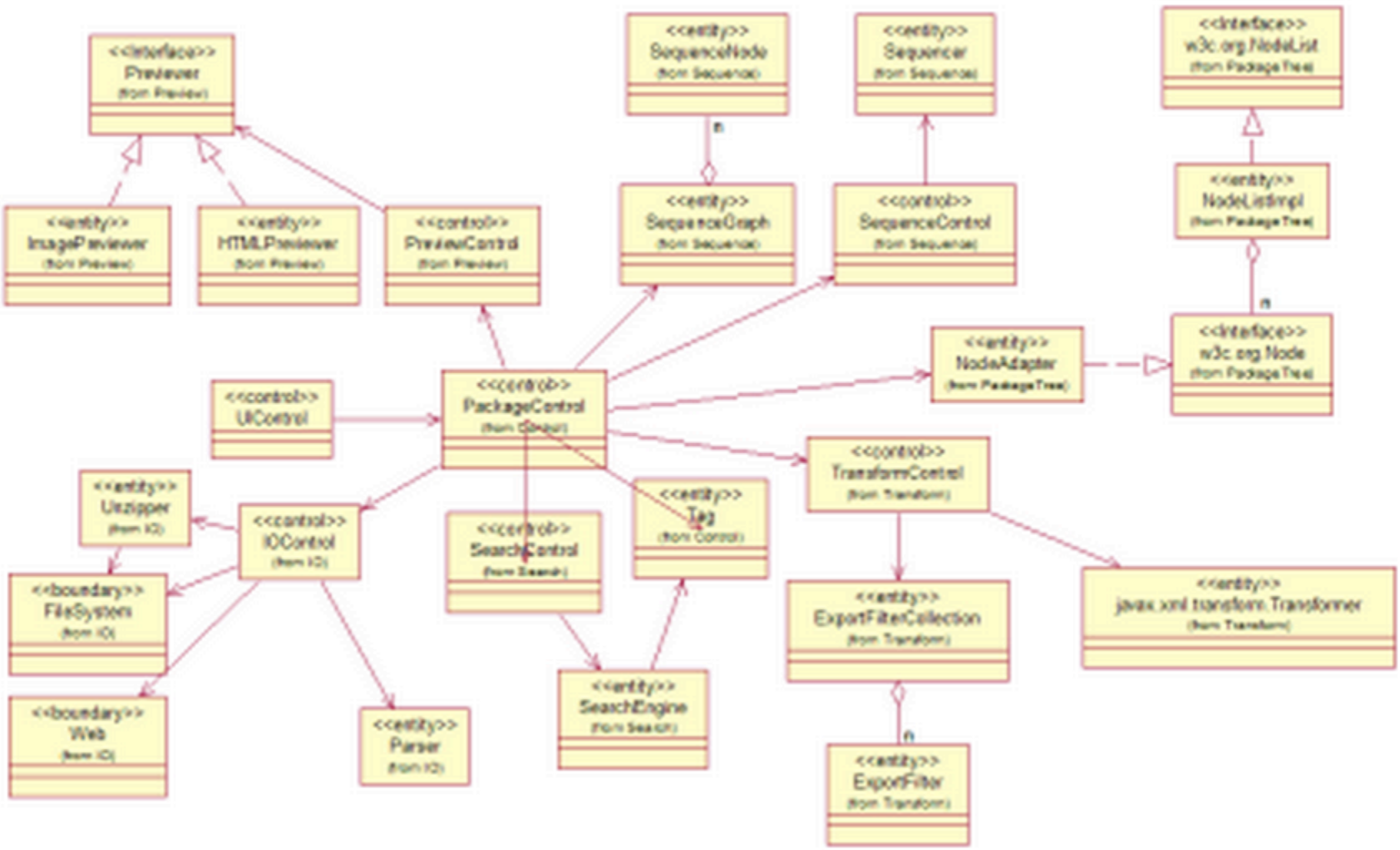
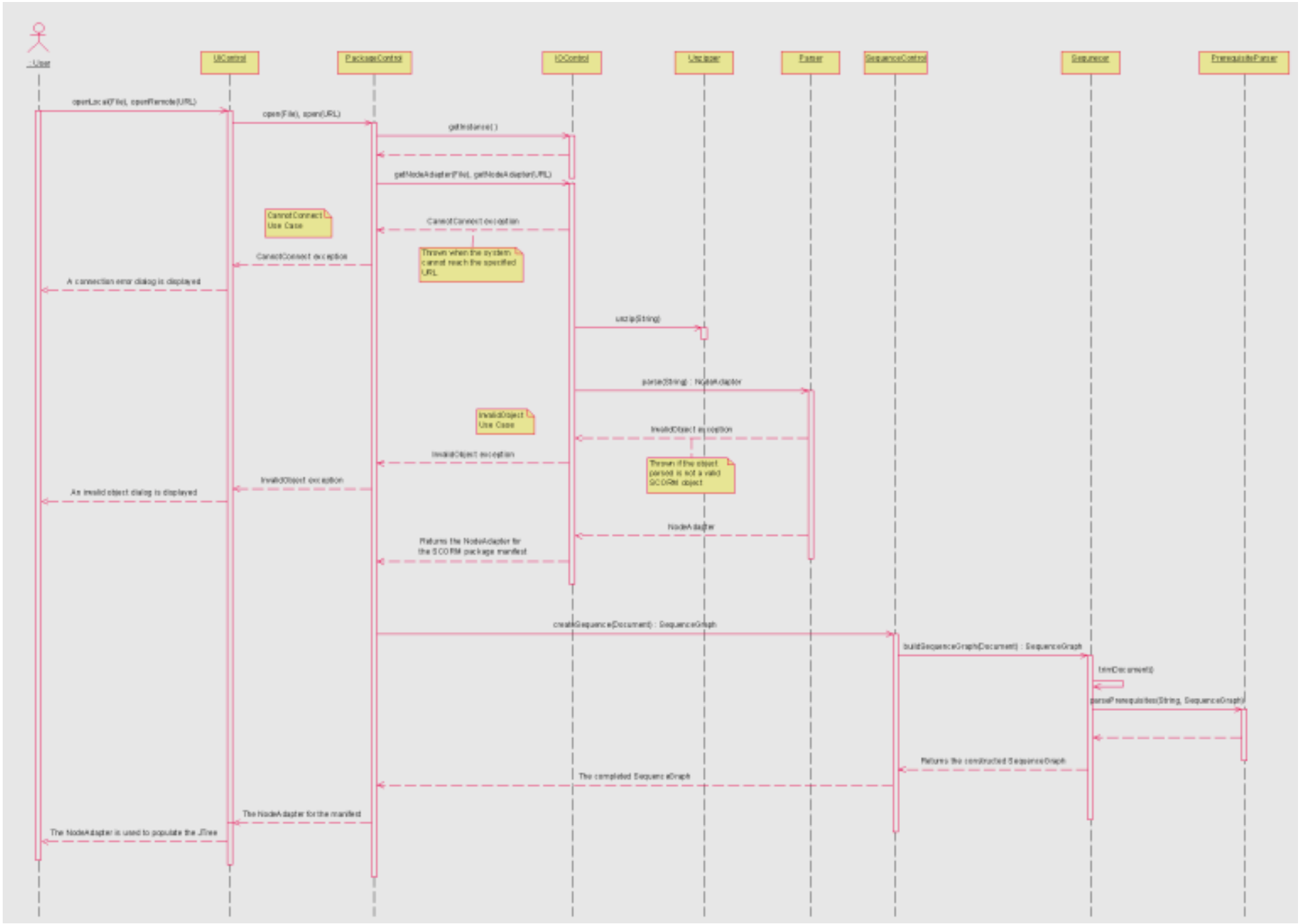
Software Visualization

Presented by Sam Davis


<<interface>>
Previewer
(from Preview)
<<entity>>
SequenceNode
(from Sequence)
<<entity>>
Sequencer
(from Sequence)
<<interface>>
w3c.org.NodeList
(from PackageTree)
<<entity>>
ImagePreviewer
(from Preview)
<<entity>>
HTMLPreviewer
(from Preview)
<<control>>
PreviewControl
(from Preview)
<<entity>>
SequenceGraph
(from Sequence)
<<control>>
SequenceControl
(from Sequence)
<<entity>>
NodeListImpl
(from PackageTree)
<<entity>>
NodeAdapter
(from PackageTree)
<<interface>>
w3c.org.Node
(from PackageTree)
<<control>>
UIControl
<<control>>
PackageControl
(from GUI)
<<control>>
TransformControl
(from Transform)
<<entity>>
Unzipper
(from IO)
<<control>>
IOControl
(from IO)
<<control>>
SearchControl
(from Search)
<<entity>>
Tag
(from Control)
<<boundary>>
FileSystem
(from IO)
<<entity>>
ExportFilterCollection
(from Transform)
<<entity>>
javax.xml.transform.Transformer
(from Transform)
<<boundary>>
Web
(from IO)
<<entity>>
Parser
(from IO)
<<entity>>
SearchEngine
(from Search)
<<entity>>
ExportFilter
(from Transform)


: User
UIControl
PackageControl
IOControl
Unzipper
Parser
SequenceControl
Sequencer
PrerequisiteParser
openLocal(File), openRemote(URL)
open(File), open(URL)
getInstance()
getNodeAdapter(File), getNodeAdapter(URL)
CannotConnect Use Case
CannotConnect exception
Thrown when the system cannot reach the specified URL
CannotConnect exception
A connection error dialog is displayed
unzip(String)
parse(String) : NodeAdapter
InvalidObject Use Case
InvalidObject exception
InvalidObject exception
Thrown if the object parsed is not a valid SCORM object
InvalidObject exception
An invalid object dialog is displayed
NodeAdapter
Returns the NodeAdapter for the SCORM package manifest
createSequence(Document) : SequenceGraph
buildSequenceGraph(Document) : SequenceGraph
trimDocument()
parsePrerequisites(String, SequenceGraph)
Returns the constructed SequenceGraph
The completed SequenceGraph
The NodeAdapter for the manifest
The NodeAdapter is used to populate the JTree

More than Just UML!

- UML is about static structure of software
- In terms of abstractions like
  - Procedures
  - Objects
  - Files
  - Packages
- But…

Software is Dynamic!

- Abstractions are for developers
- Users care about behaviour
- Visualize behaviour of software at run time
  - Find errors
  - Find performance bottlenecks

What can we visualize?

Test Results

- Hundreds, maybe thousands of tests
- For each test:
  - Purpose
  - Result (pass or fail)
    - Could be per-configuration or per-version
  - Relevant parts of the code

Detailed Execution Data

- Could be for many executions
- Dynamic events as opposed to summary data

Summary Data: Examples

- Total running time
- Number of times a method was called
- Amount of time CPU was idle

Dynamic Events: Examples

- Memory allocation
- System calls
- Cache misses
- Page faults
- Pipeline flushes
- Process scheduling
- Completion of disk reads or writes
- Message receipt
- Application phases

Really Detailed Execution Data

- Logging virtual machines can capture *everything*
  - Enough data to replay program execution and recreate the entire machine state at any point in time
  - Allows “time-traveling”
  - For long running systems, data could span months
- Uses:
  - Debugging
  - Understanding attacks

Strata_Various: Multi_Layer Visualization of Dynamics in Software System Behavior

Doug Kimelman, Bryan Rosenburg, Tova Roth

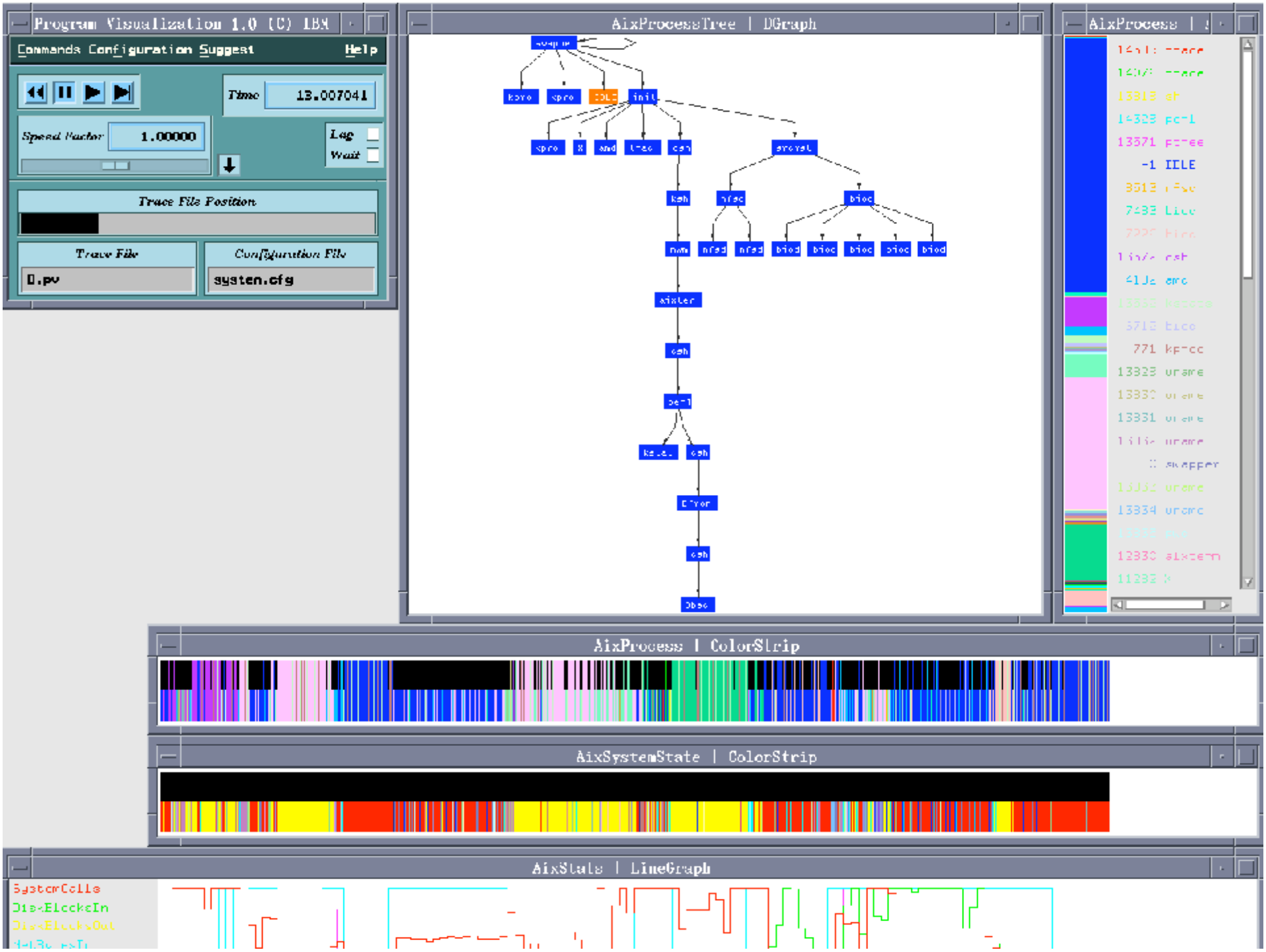
Proc. Fifth IEEE Conf. Visualization ’94, IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 172–178.

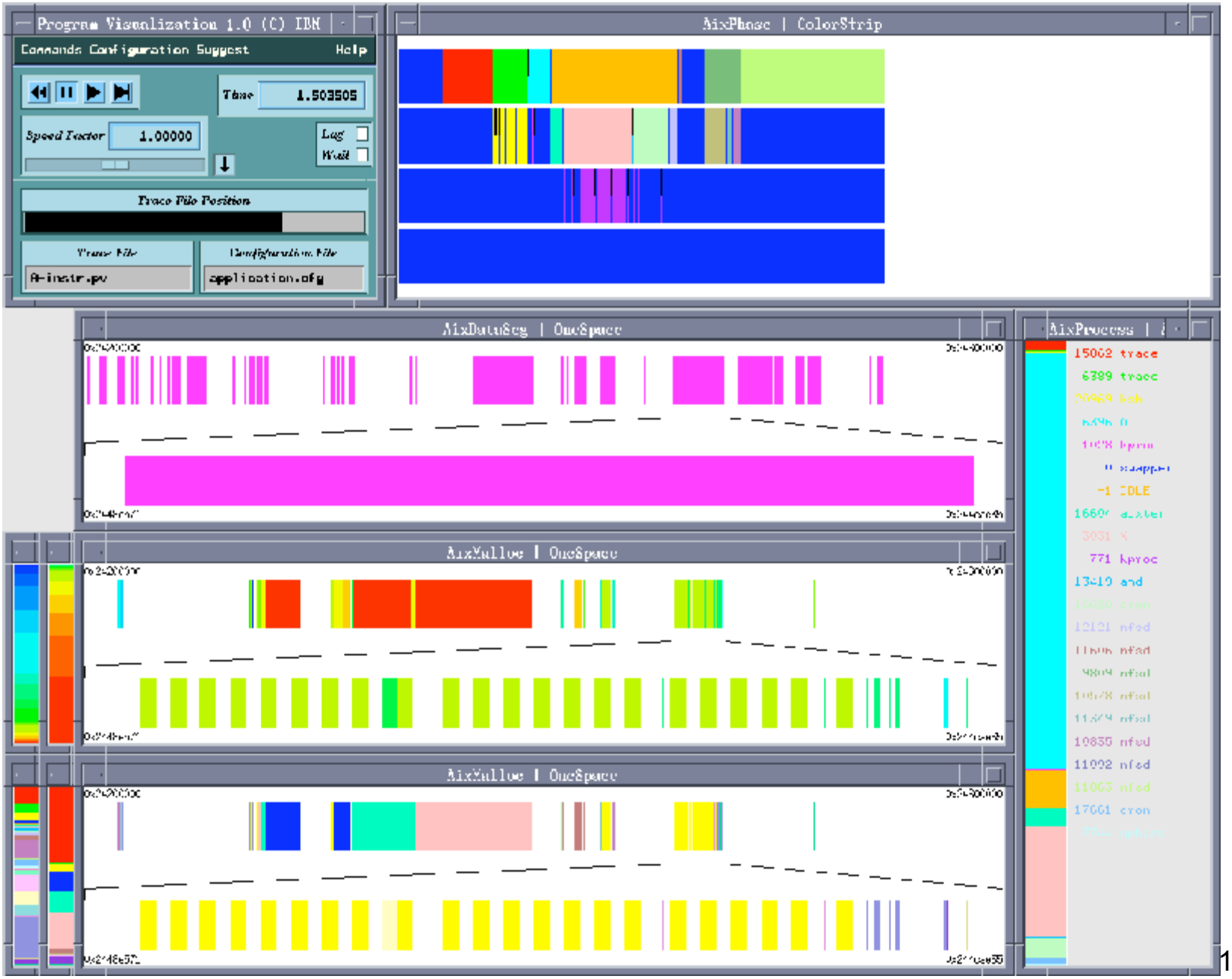
Strata_Various

- Trace-driven program visualization
- Trace: sequence of <time, event> pairs
- Events captured from all layers:
  - Hardware
  - Operating System
  - Application
- Replay execution history
- Coordinate navigation of event views

Strata_Various: Main Argument

- Debugging and tuning requires simultaneously analyzing behaviour at multiple layers of the system


Program Visualization 1.0 (C) IBM
Commands Configuration Suggest Help
Time 13.007041
Speed Factor 1.00000
Lag
Wait
Trace File Position
Trace File
0.pv
Configuration File
system.cfg
AixProcessTree | DGraph
AixProcess | ColorStrip
AixSystemState | ColorStrip
AixState | LineGraph
SystemCalls
DiskBlocksIn
DiskBlocksOut


Program Visualization 1.0 (C) IBM
Commands Configuration Suggest Help
Time 1.503505
Speed Factor 1.00000
Log
Wait
Trace File Position
Trace File
A-instr.pv
Configuration File
application.cfg
AixPhase | ColorStrip
AixDataSeg | OneSpace
AixProcess
15002 trace
6389 trace
-1 IDLE
771 kproc
13419 and
10835 nfsd
11092 nfsd
17661 cron
AixMallwe | OneSpace
AixMallwe | OneSpace


Program Visualization 1.0 (C) IBM
Commands Configuration Suggest Help
Time 1.341370
Speed Factor 0.01000
Log
Wait
Trace File Position
Trace File
T.pv
Configuration File
pmon-sro.cfg
Aix
AixPhaseLine | SourceCode
AixProfile | OneSpace
AixProcess | ColorStrip
AixPhase | ColorStrip
RS2Pmc | Scale | LineGraph

Strata_Various: Critique

- Examples demonstrate usefulness
- Fundamentally, a good idea
  - Increasing importance as multi-core machines become standard
- Many windows
  - Titles not meaningful
  - Virtual reality cop-out
- Dubious claim that tracing does not alter behaviour

SeeSoft

- Zoomed out view of source code
  - Lines of code displayed as thin horizontal lines
  - Preserve indentation, length
  - Can colour lines according to data
- Link with readable view of code
- Allows tying data to source code

Stephen G. Eick, Joseph L. Steffen and Eric E. Sumner, Jr. “SeeSoft – A Tool for Visualizing Line-Oriented Software Statistics.” *IEEE Transactions on Software Engineering*, 18(11):957-968, November 1992.

SeeSoft Example

Visually Encoding Program Test Information to Find Faults in Software

(Tarantula)

James Eagan, Mary Jen Harrold, James A. Jones, and John Stasko, Proc. InfoVis 2001 pp. 33-36.

Tarantula

- Extends SeeSoft idea
- Defines colour mapping for LOC based on test results
- Goal: use test results to identify broken code

Tarantula

- Input:
  - For each test:
    - Test number
    - Result (pass or fail)
    - Test coverage (list of line numbers)

Tarantula: Discrete Colour Mapping

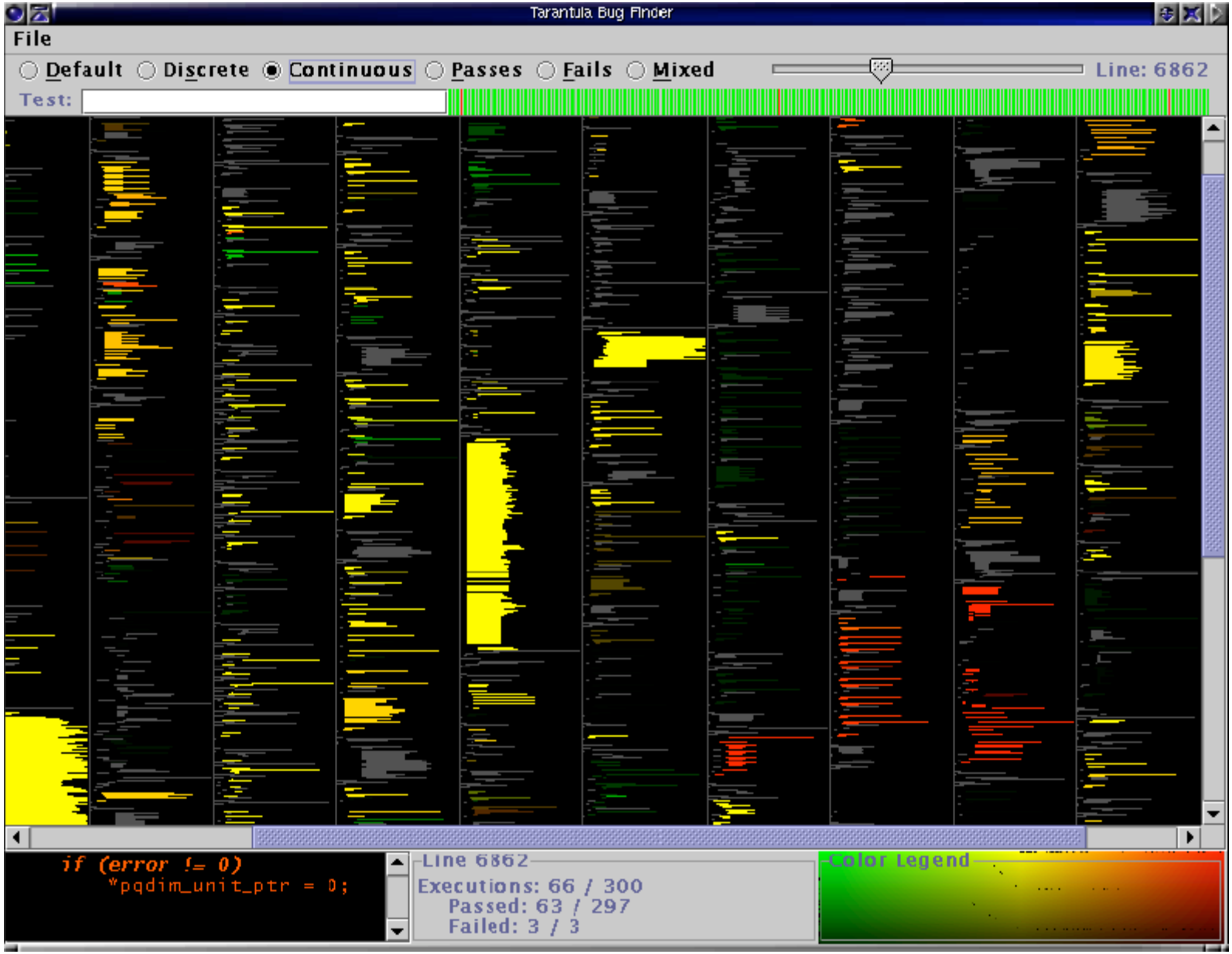
- Based on user tests
- Black background
- Colour each line
  - Red if executed by failed tests
  - Green if executed by passed tests
  - Yellow if executed by both

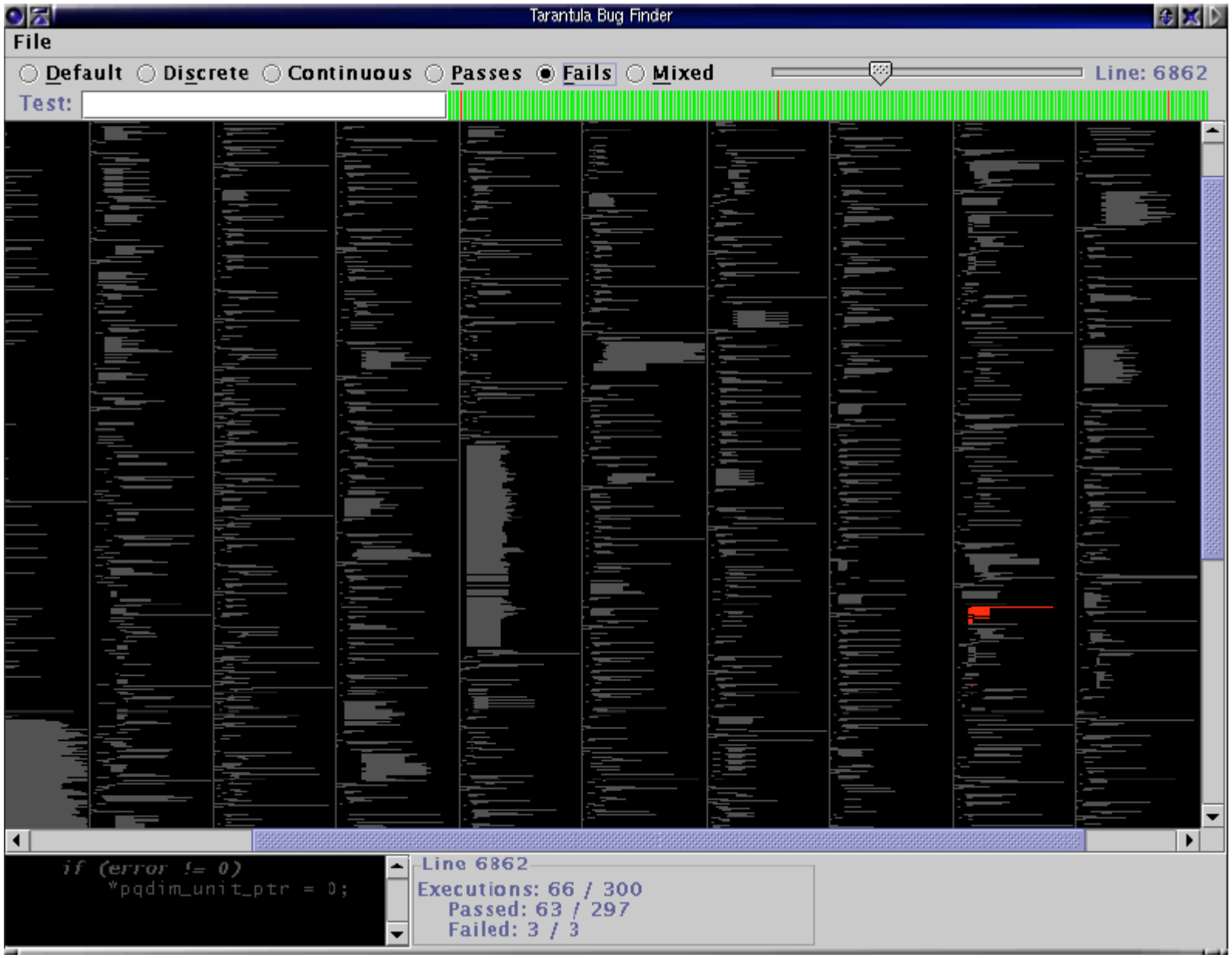
Tarantula: Continuous Colour Mapping

- Extend discrete colour mapping by
  - Interpolating between red and green
  - Adjusting brightness according to number of tests
- Possibilities:
  - Number of passed or failed tests
  - Ratio of passed to failed tests
  - Ratio of % passed to % failed

Tarantula: Continuous Colour Mapping

- For each line L
  - Let p and f be the percentages of passed and failed tests that executed L
  - If p = f = 0, colour L grey
  - Else, colour L according to
    - Hue: p / (p + f), where 0 is red and 1 is green
    - Brightness: max(p, f)


Tarantula Bug Finder
File
Default
Discrete
Continuous
Passes
Fails
Mixed
Line: 6862
Test:
if (error != 0)
*pqdim_unit_ptr = 0;
Line 6862
Executions: 66 / 300
Passed: 63 / 297
Failed: 3 / 3
Color Legend


Tarantula Bug Finder
File
Default
Discrete
Continuous
Passes
Fails
Mixed
Line: 6862
Test:
if (error != 0)
*pqdim_unit_ptr = 0;
Line 6862
Executions: 66 / 300
Passed: 63 / 297
Failed: 3 / 3

Tarantula: Critique

- Visualizing test results could be useful, this is a first step
- Future work: does colouring help to find broken code?
- Colouring: simple idea made complex
- Tests identified only by number
  - Better: name tests
  - Better still: can we visualize the *meaning* of tests?

Visualization of Program-Execution Data for Deployed Software

(Gammatella)

Alessandro Orso, James Jones, and Mary Jean Harrold.

Proc. of the ACM Symp. on Software Visualization, San Diego, CA, June 2003, pages 67--76.

Gammatella

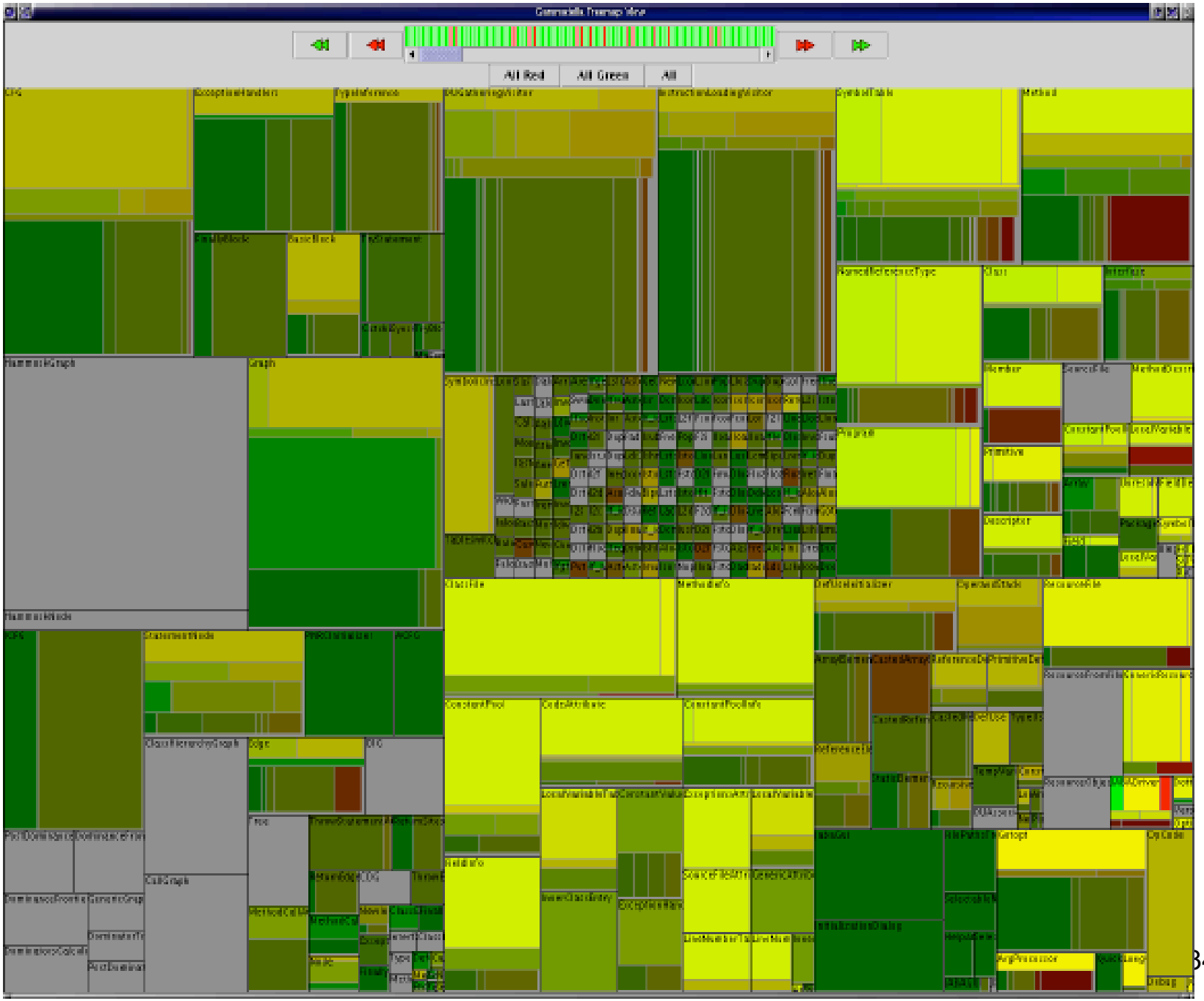
- Collection and storage of program-execution data
- Visualization of data about many executions

Gammatella: Executions

- Code coverage and profiling data
- Execution properties
  - OS
  - Java version
  - Etc.
- Filters
  - Boolean predicate logic
- Summarizers

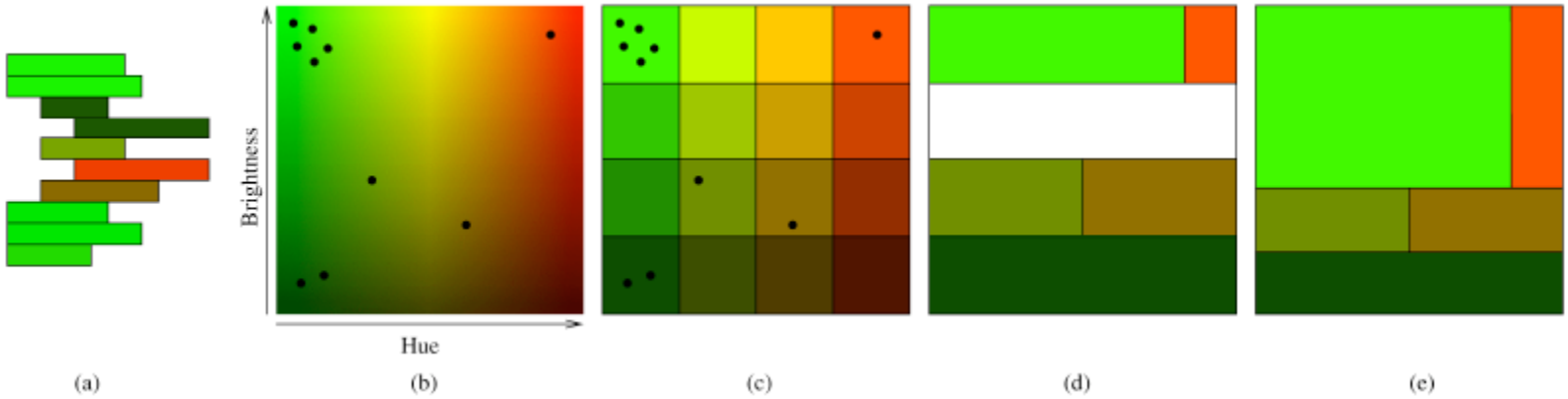
Gammatella: Coloured, Tri-Level Representation

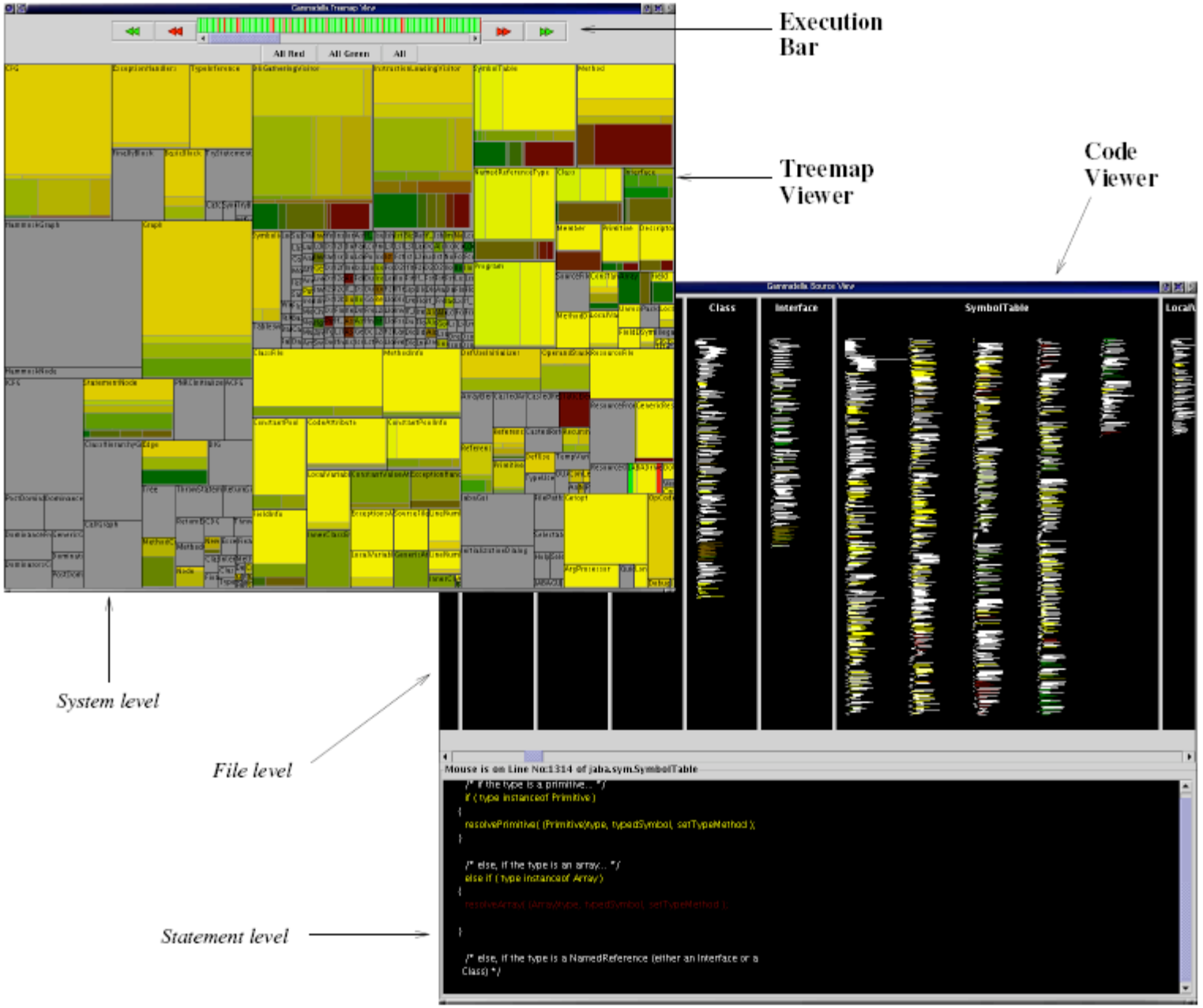
- System level
  - Treemap of package/class hierarchy
- File level:
  - SeeSoft-like view of code
- Statement level:
  - Source code (coloured text)
- Colours based on exceptions
  - Other colourings possible, e.g. profiling data


All Red
All Green
All

One Level Treemap

- Layout algorithm for treemap of depth 1
  - Preserves relative placement of colours




Execution Bar
Treemap Viewer
Code Viewer
All Red
All Green
All
Class
Interface
SymbolTable
System level
File level
Statement level
Mouse is on Line No:1314 of jaba.sym.SymbolTable

Gammatella: Critique

- Complete system – not just a visualization
- Effectively links code to structure
- Trial usage discovered useful but high-level information
  - Mainly relied on system view
  - Would be nice to see examples using file and statement level views

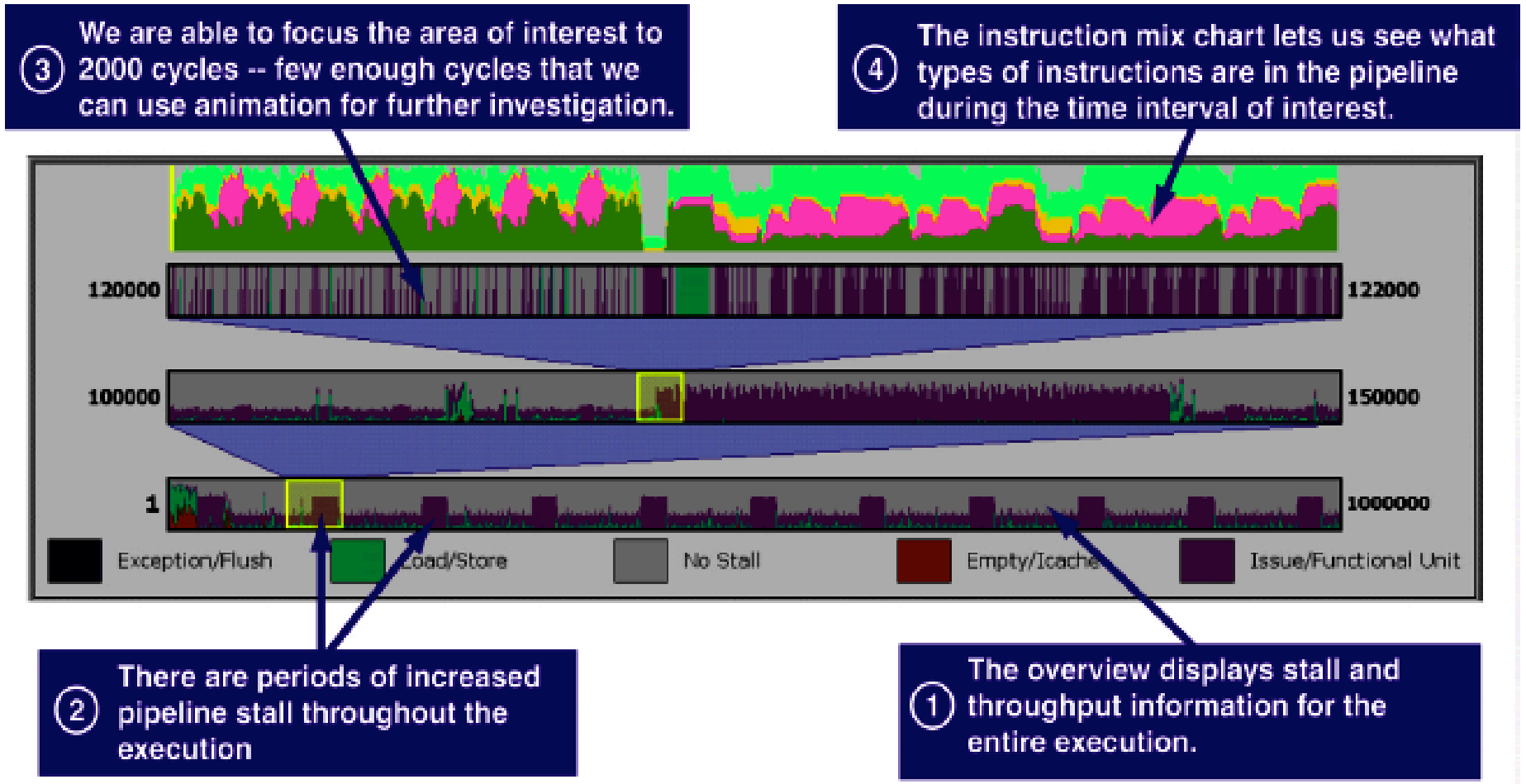
Visualizing Application Behavior on Superscalar Processors

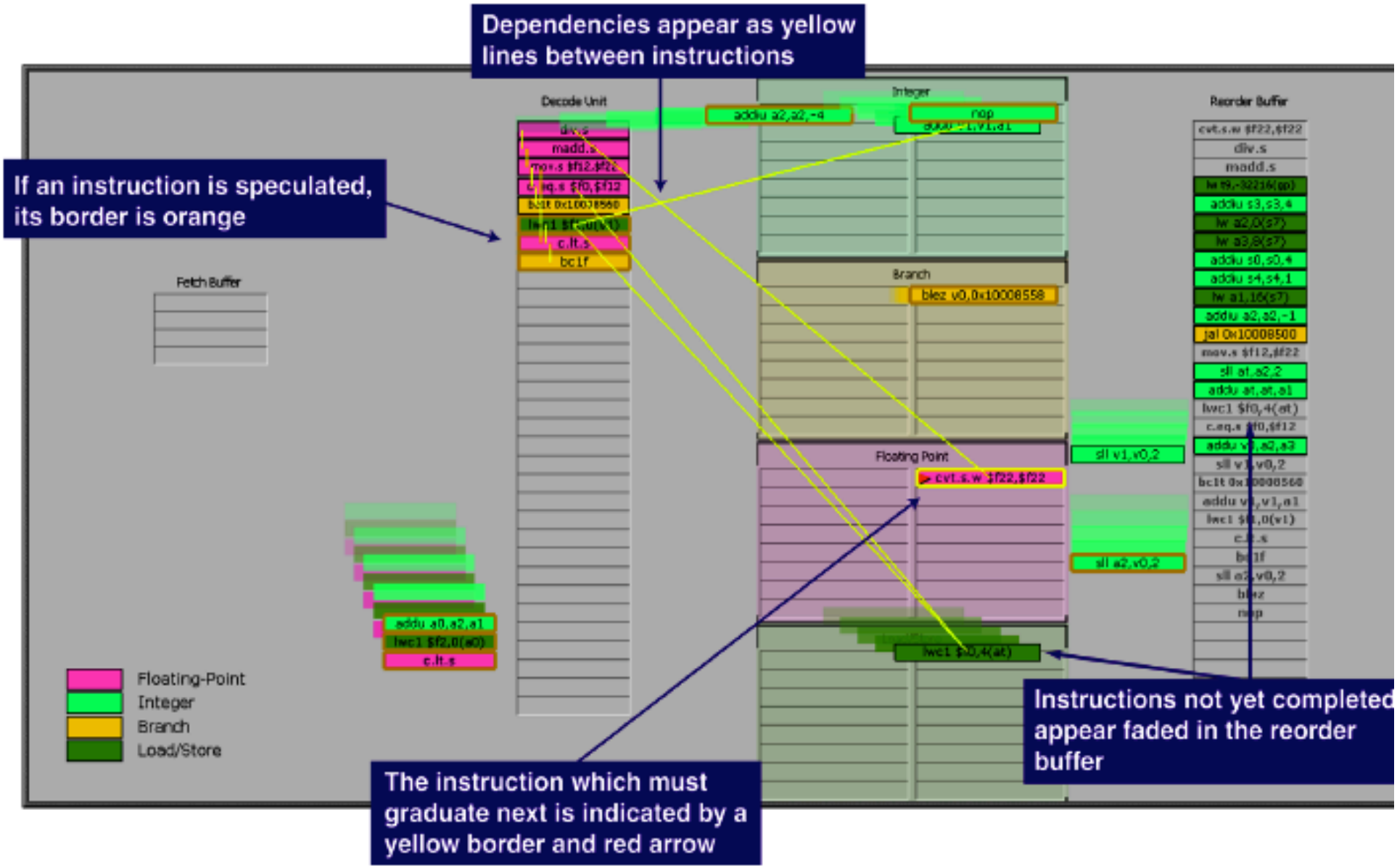
Chris Stolte, Robert Bosch, Pat Hanrahan, and Mendel Rosenblum

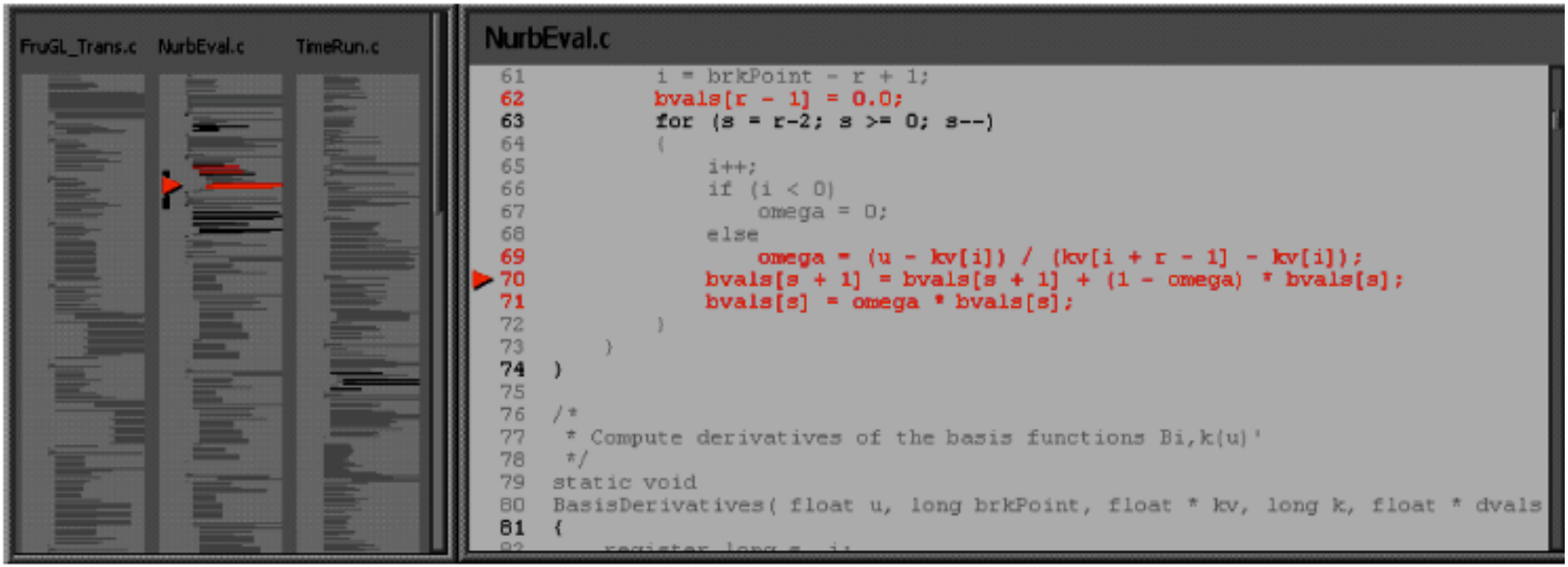
Proc. InfoVis 1999

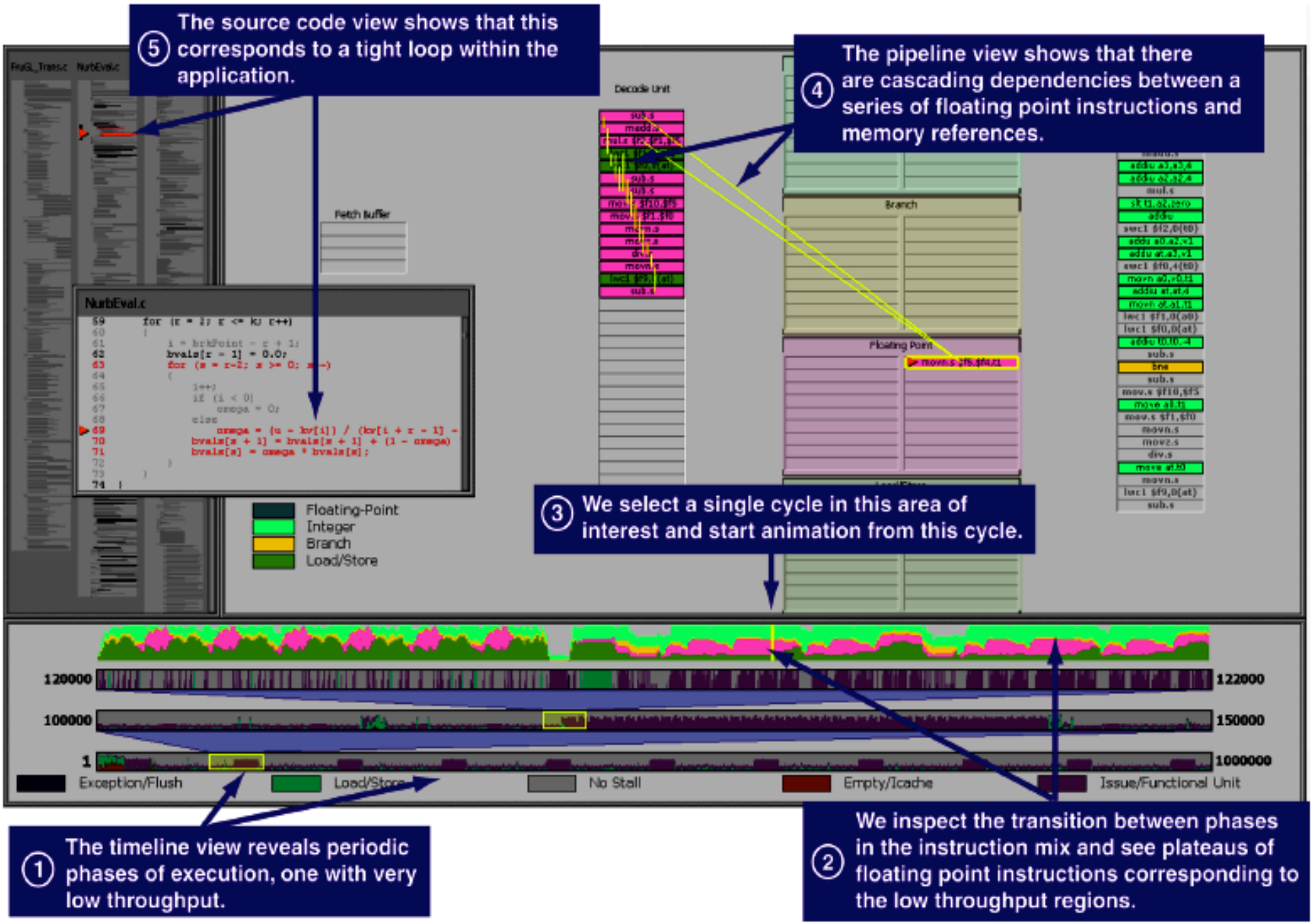
Superscalar Processors: Quick Overview

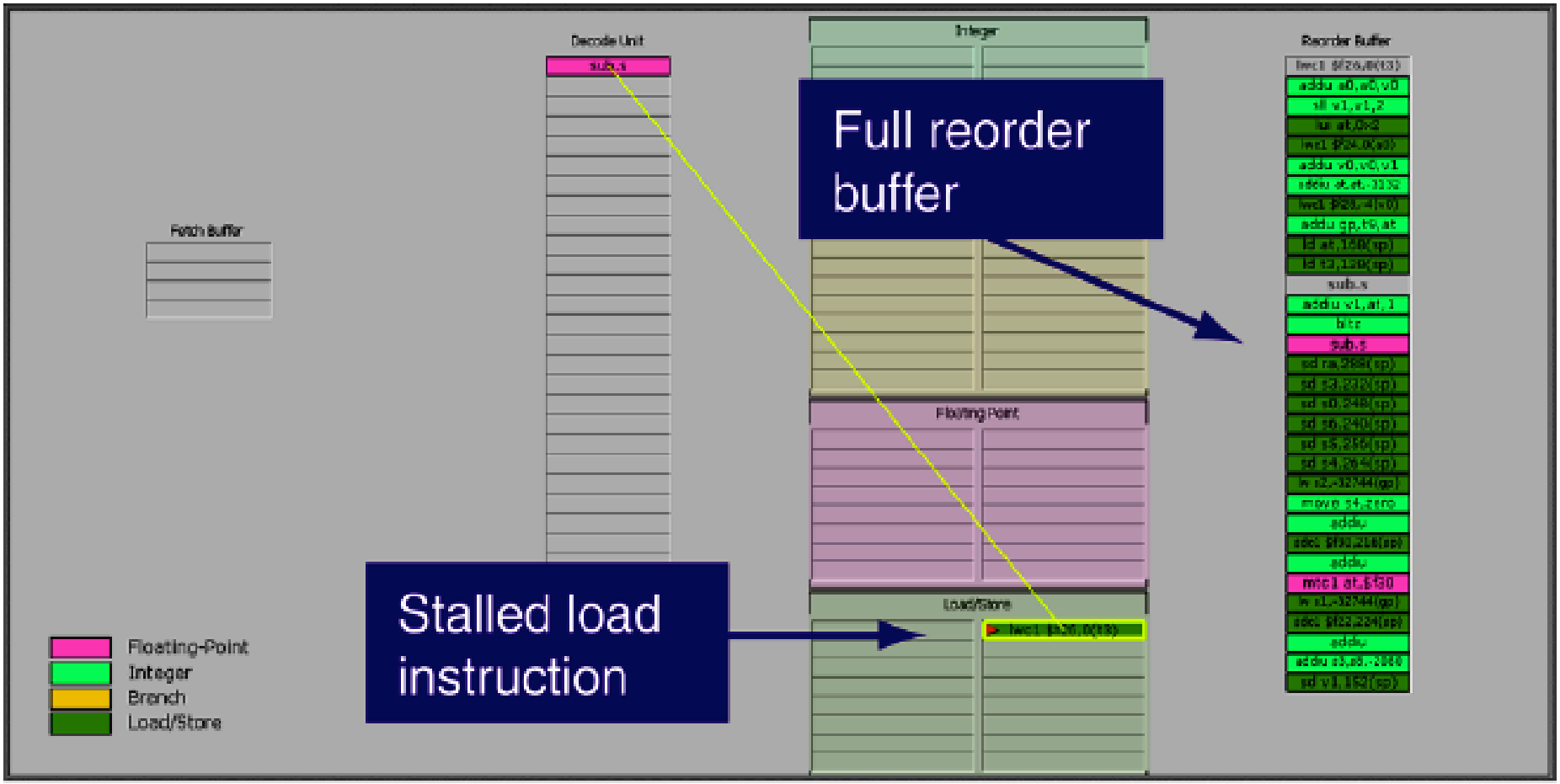
- Pipeline
- Multiple Functional Units
  - Instruction-Level Parallelism (ILP)
- Instruction Reordering
- Branch Prediction and Speculation
- Reorder Buffer
  - Instructions wait to graduate (exit pipeline)


3 We are able to focus the area of interest to 2000 cycles -- few enough cycles that we can use animation for further investigation.
4 The instruction mix chart lets us see what types of instructions are in the pipeline during the time interval of interest.
120000
122000
100000
150000
1
1000000
Exception/Flush
Load/Store
No Stall
Empty/Icache
Issue/Functional Unit
2 There are periods of increased pipeline stall throughout the execution
1 The overview displays stall and throughput information for the entire execution.

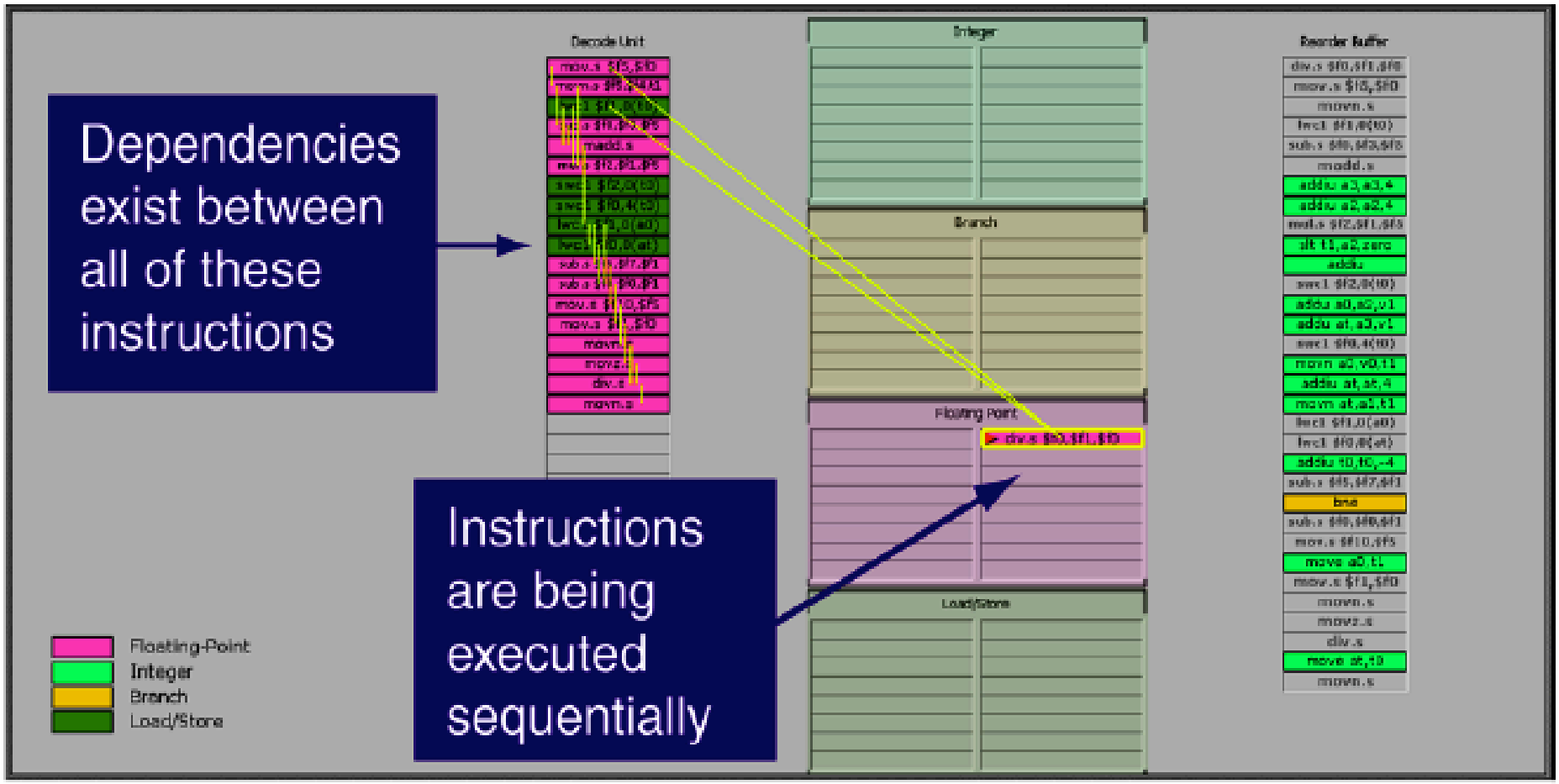

Dependencies appear as yellow lines between instructions
If an instruction is speculated, its border is orange
Decode Unit
Integer
Reorder Buffer
Fetch Buffer
Branch
Floating Point
addu a2,a2,-4
nop
div.s
madd.s
bc1f
blez v0,0x10008558
cvt.s.w $f22,$f22
sll v1,v0,2
sll a2,v0,2
addu a0,a2,a1
lwc1 $f2,0(a0)
c.lt.s
lwc1 $f0,4(at)
Floating-Point
Integer
Branch
Load/Store
The instruction which must graduate next is indicated by a yellow border and red arrow
Instructions not yet completed appear faded in the reorder buffer


FruGL_Trans.c
NurbEval.c
TimeRun.c
NurbEval.c
61 i = brkPoint - r + 1;
62 bvals[r - 1] = 0.0;
63 for (s = r-2; s >= 0; s--)
64 {
65 i++;
66 if (i < 0)
67 omega = 0;
68 else
69 omega = (u - kv[i]) / (kv[i + r - 1] - kv[i]);
70 bvals[s + 1] = bvals[s + 1] + (1 - omega) * bvals[s];
71 bvals[s] = omega * bvals[s];
72 }
73 }
74)
75
76 /*
77 * Compute derivatives of the basis functions Bi,k(u)'
78 */
79 static void
80 BasisDerivatives(float u, long brkPoint, float * kv, long k, float * dvals
81 {

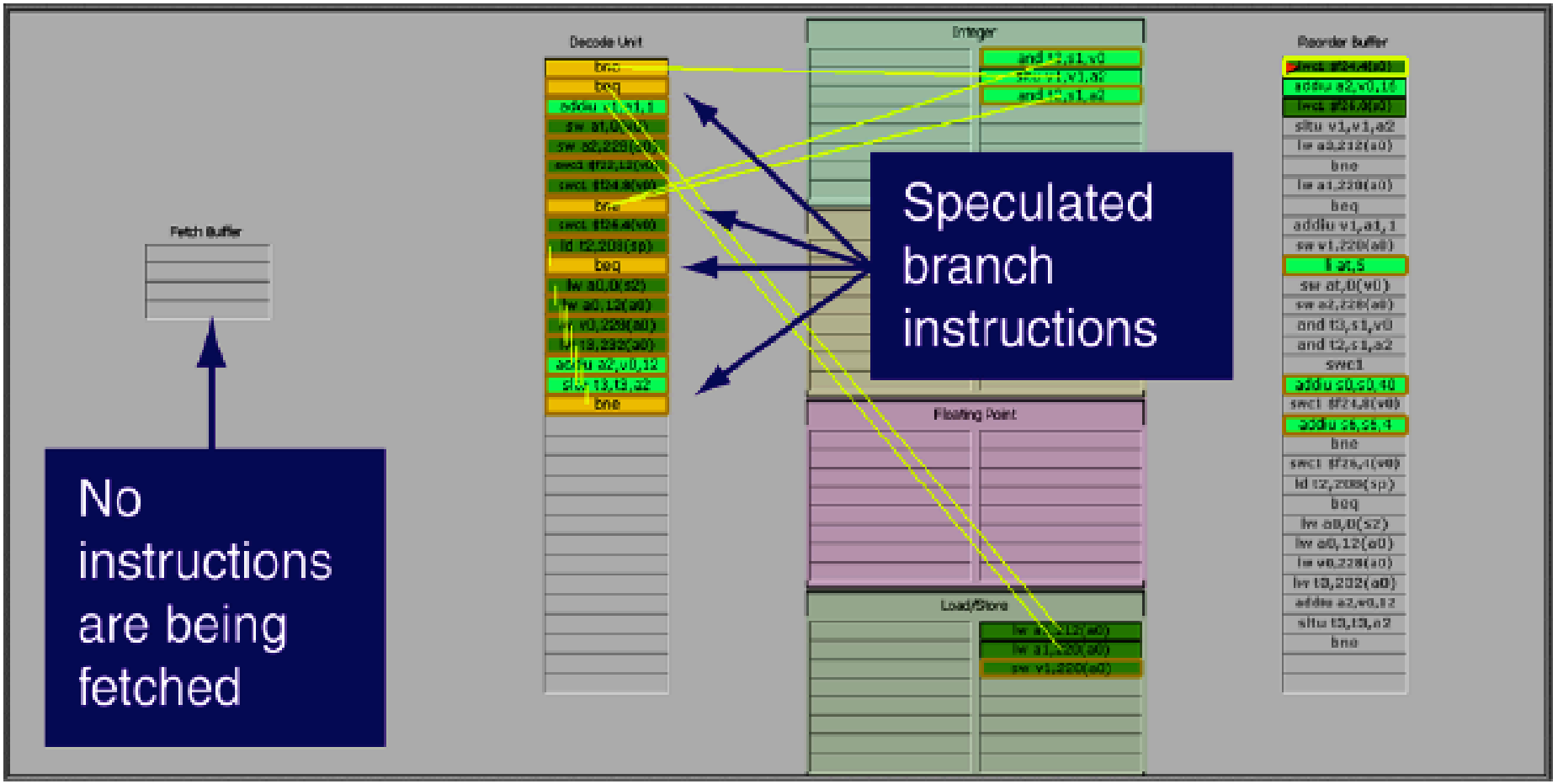

5 The source code view shows that this corresponds to a tight loop within the application.
4 The pipeline view shows that there are cascading dependencies between a series of floating point instructions and memory references.
3 We select a single cycle in this area of interest and start animation from this cycle.
1 The timeline view reveals periodic phases of execution, one with very low throughput.
2 We inspect the transition between phases in the instruction mix and see plateaus of floating point instructions corresponding to the low throughput regions.
Decode Unit
Fetch Buffer
Branch
Floating Point
NurbEval.c
Floating-Point
Integer
Branch
Load/Store
120000
100000
1
122000
150000
1000000
Exception/Flush
Load/Store
No Stall
Empty/Icache
Issue/Functional Unit



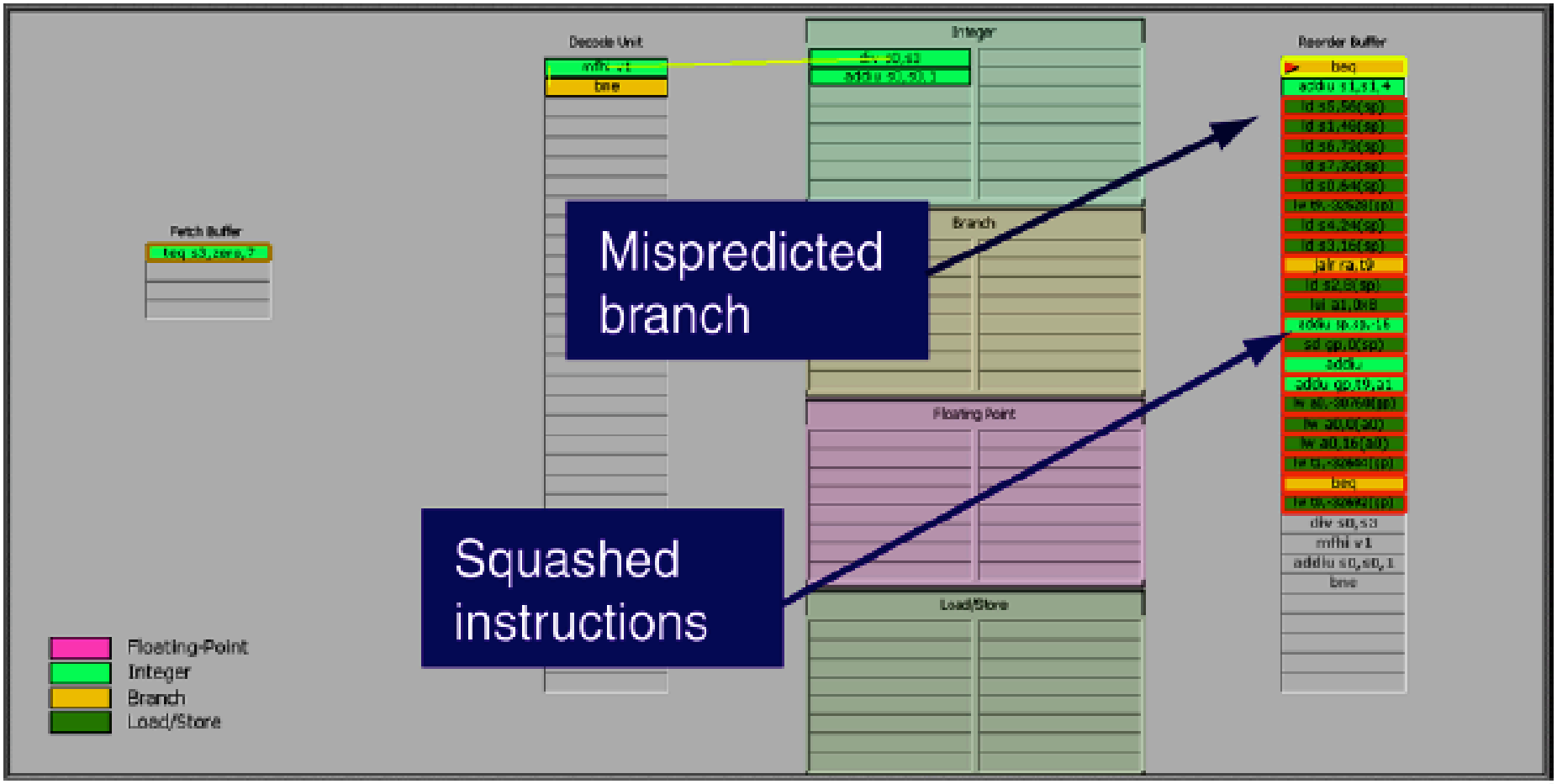
Memory stall



Dependencies



Deep speculation



Branch misprediction

Critique

- Most code doesn't need this level of optimization, but
  - The visualization is effective, and would be useful for code that does
  - May reduce the expertise needed to perform low level optimzation
- Might be effective as a teaching tool
- Bad color scheme: black/purple/brown
- Does it scale with processor complexity?

Papers

- D. Kimelman, B. Rosenburg, and T. Roth, “Strata-Various: Multi-Layer Visualization of Dynamics in Software System Behavior,” Proc. Fifth IEEE Conf. Visualization ’94, IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 172–178.

- James Eagan, Mary Jen Harrold, James A. Jones, and John Stasko, "Visually Encoding Program Test Information to Find Faults in Software." Proc. InfoVis 2001 pp. 33-36.

Papers

- Alessandro Orso, James Jones, and Mary Jean Harrold. "Visualization of Program-Execution Data for Deployed Software." Proc. of the ACM Symp. on Software Visualization, San Diego, CA, June 2003, pages 67--76.

- Chris Stolte, Robert Bosch, Pat Hanrahan, and Mendel Rosenblum, "Visualizing Application Behavior on Superscalar Processors." Proc. InfoVis 1999