Software Visualizations

Rolf Biehn

What is Software Visualization?

- Visualization of a software systems based on their structure, history, or behavior
- Today's presentation:
  - Program Execution Traces
  - Source Code History
  - Program Optimization

Execution Patterns in Object-Oriented Visualization

- David Lorenz et al.

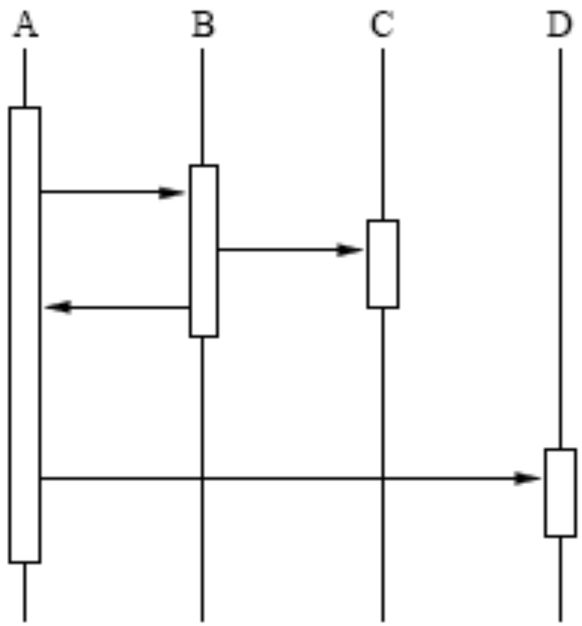
What is it?

- Techniques to visualize the execution flow and execution patterns
- Input is call traces from instrumented code

Motivation

- Understand program execution flow in order to program or debug it

UML Visualization



UML

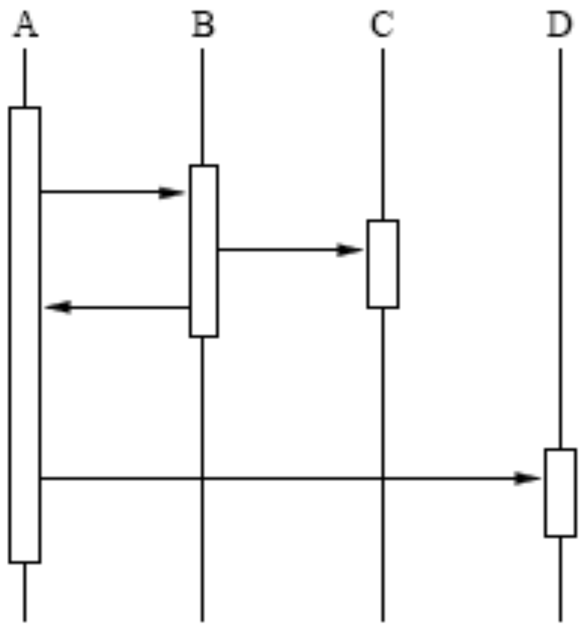
UML Discussion

+Scales better than directed graphs

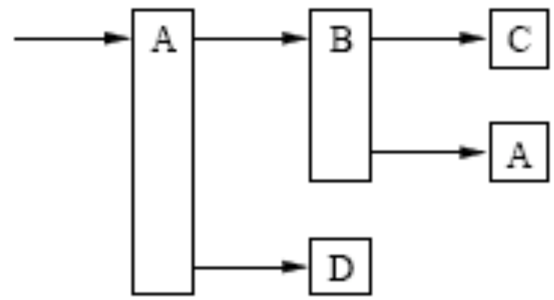
-Vertical Space is consumed quickly

-Somewhat difficult to read

UML Visualization



UML



Call Graph Tree

Execution Pattern Discussion

+Easier to read than an UML diagram (no “bouncing between axis”)

+Horizontal & Vertical space is used more efficiently

+Enables better user interaction

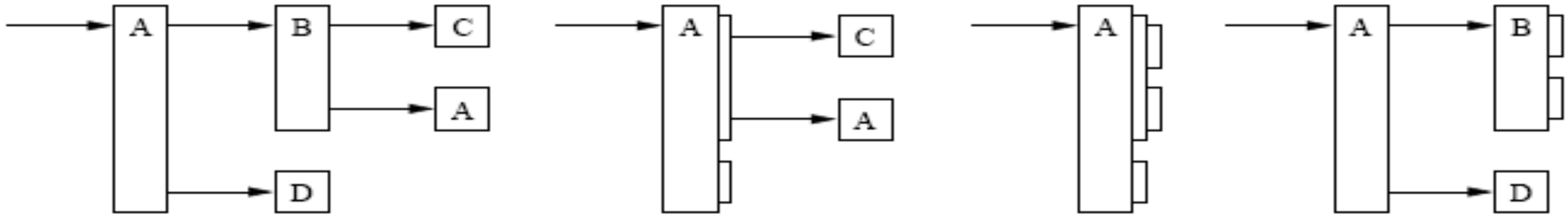


Figure 9: Schematic view of flattening

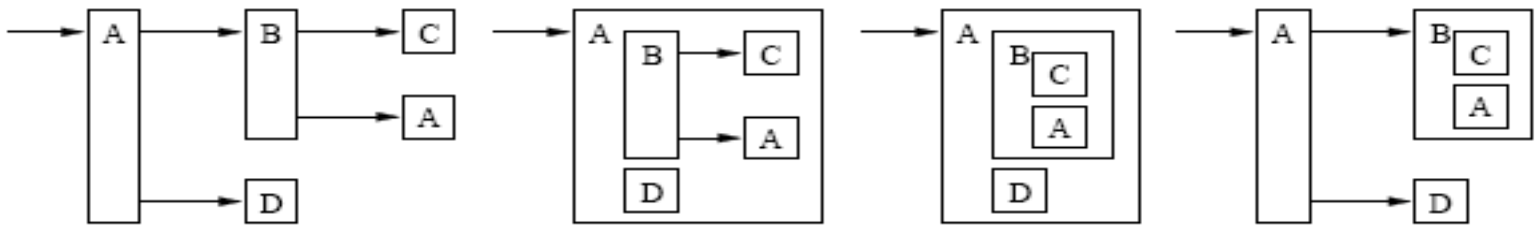
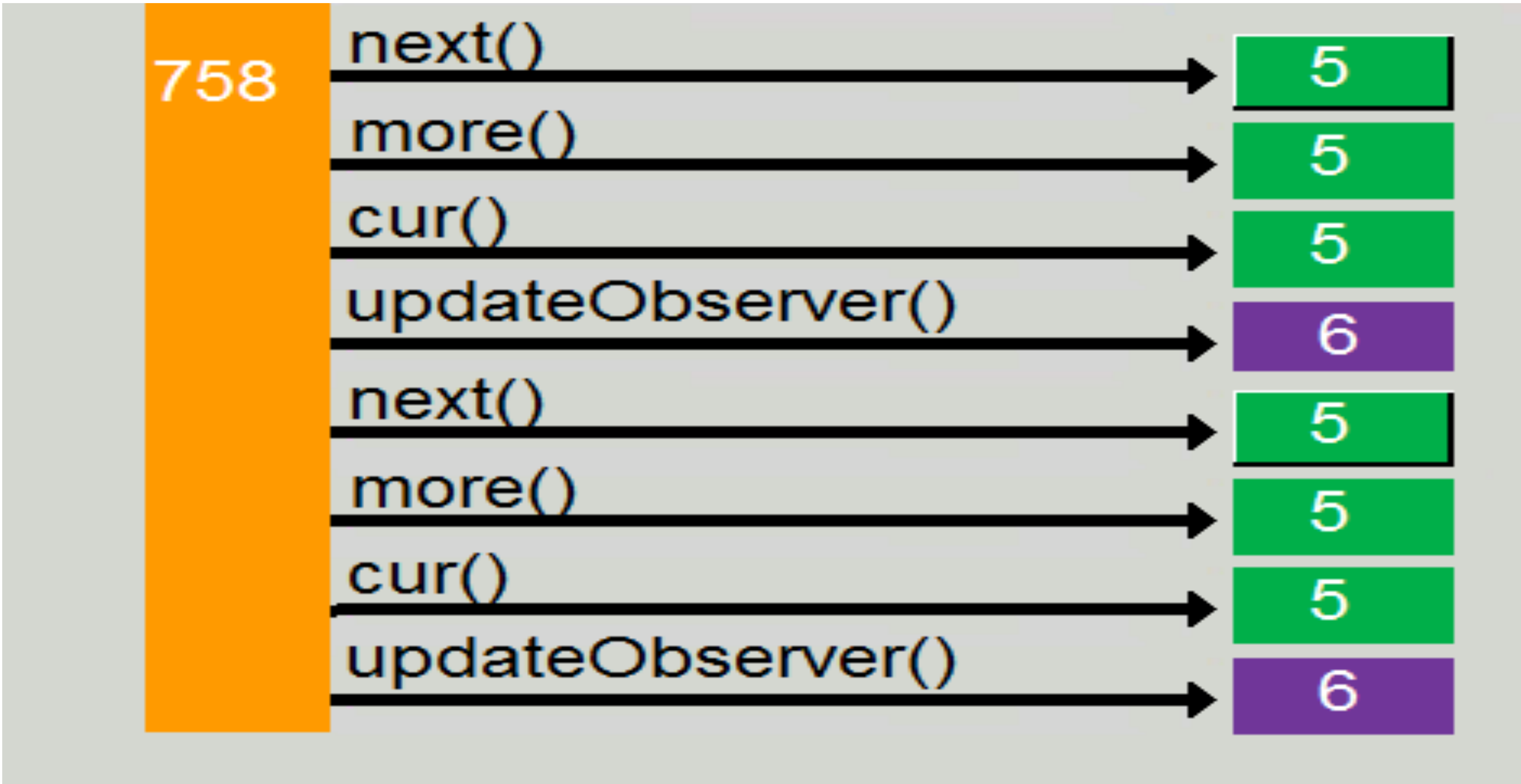


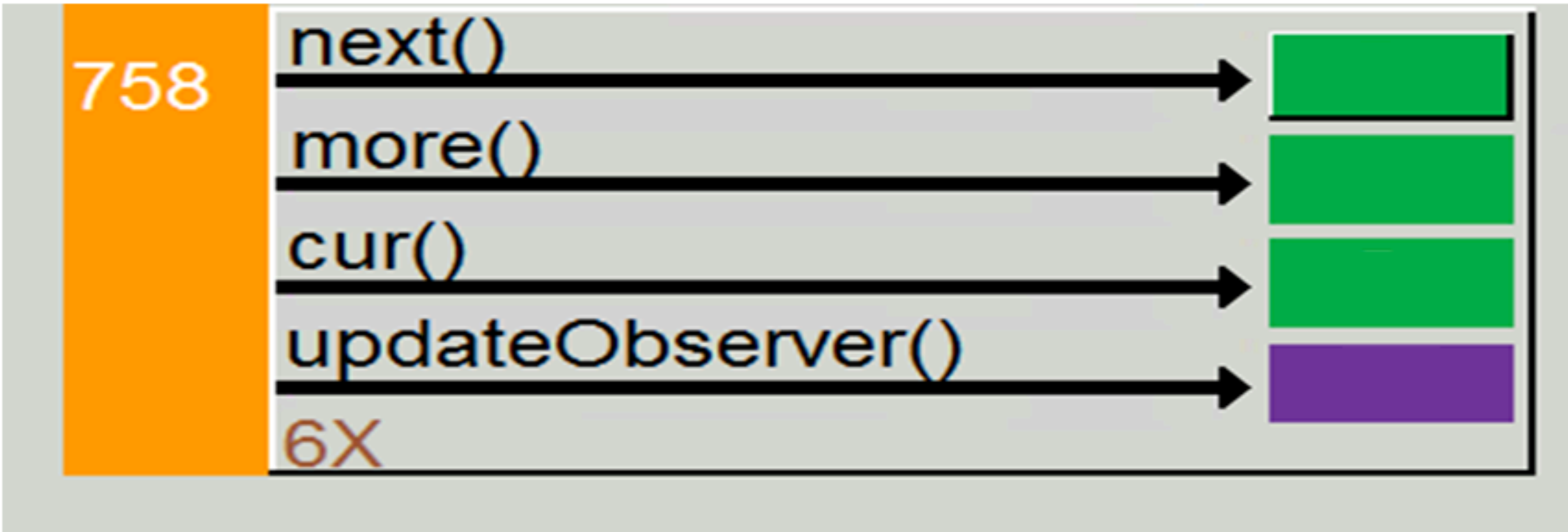
Figure 10: Schematic view of underlaying

- Flattening is useful for System libraries
- Can collapse and expand nodes
- Can search & filter (with expressions)
- Panning & Zooming also supported

next()
more() const
curr() const
update(Observable*)
767
next()
more() const
curr() const



- 3D box indicates a collapsed node
- Colors correspond to a class
- ID #s represent identity of the object



- 3D box used to show pattern
- Saves lots of space in call traces
- Can expand/contract
- Number (6X) shows number of repetitions
- Also applies to recursion

How to detect pattern?

- Bunch of dimension:
  - Identity, Class Identity, Message Structure, Depth Limiting, Repetition, Polymorphism, Associatively, Commutatively
- Create a hash function for each leaf node which considers these dimensions
- Create a recursive hash function which considers its children in the call graph
- Put all nodes into a dictionary
- How long does it take? Memory concerns?

Evaluation

- Understand program execution flow in order to program or debug it
  - (B) Looks like it should work, if implemented carefully
  - How to navigate from high-level if I don't know precisely what I want to see?
  - What about multi-threading?
  - How well does it scale? What if number of Classes exceeds distinguishable colours?

CVSscan: Visualization of Code Evolution

- Alex Telea, et al.

What is it?

- CVSScan is part of a larger suite of tools called Visual Code Navigator
- Provides information of the history of check-ins

Motivation

- Answer the following questions
  - Who performed these modifications of the code
  - Which parts of the code are unstable?
  - How are changes correlated?
  - How are the development tasks distributed?
  - What is the context in which a piece of code appeared?

Dimensions to Show

- All encoded using colors
  - Author
  - Content (block, comment, references)
  - Evolution (add/remove/delete/unchanged)

Global Line Position

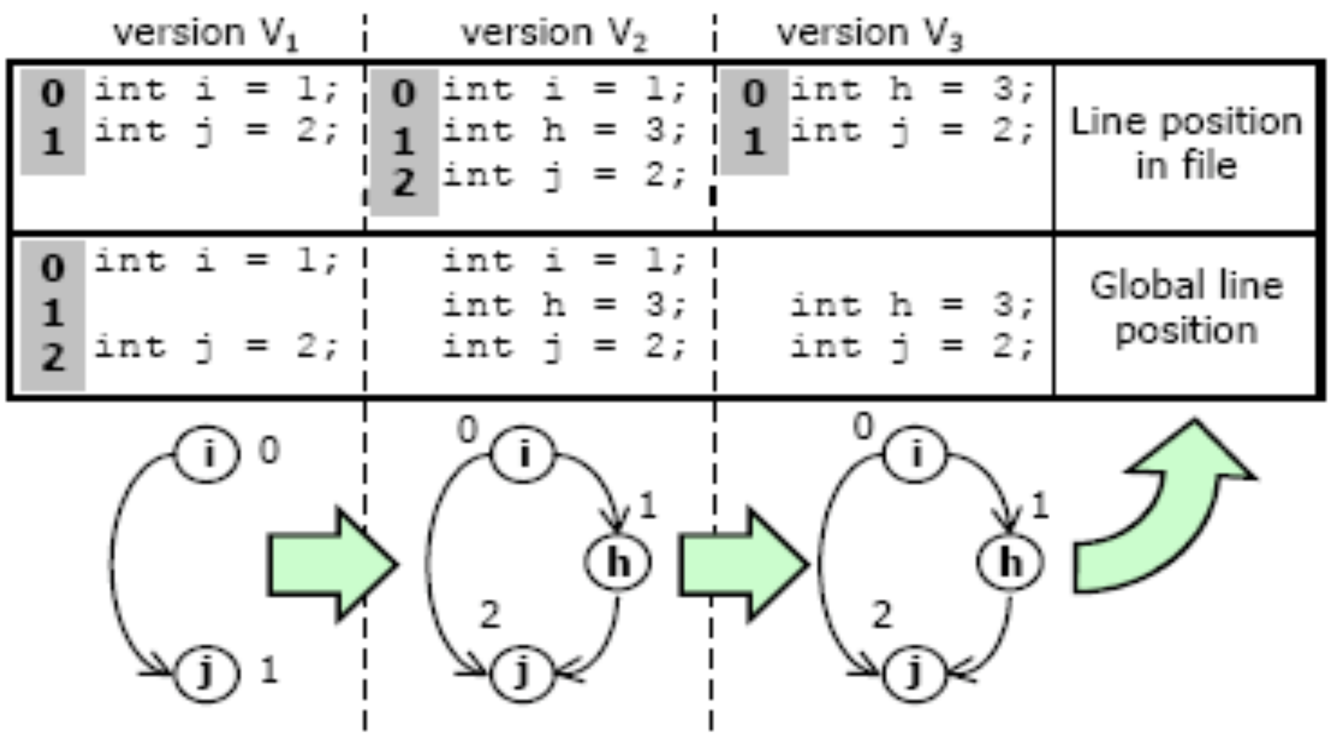
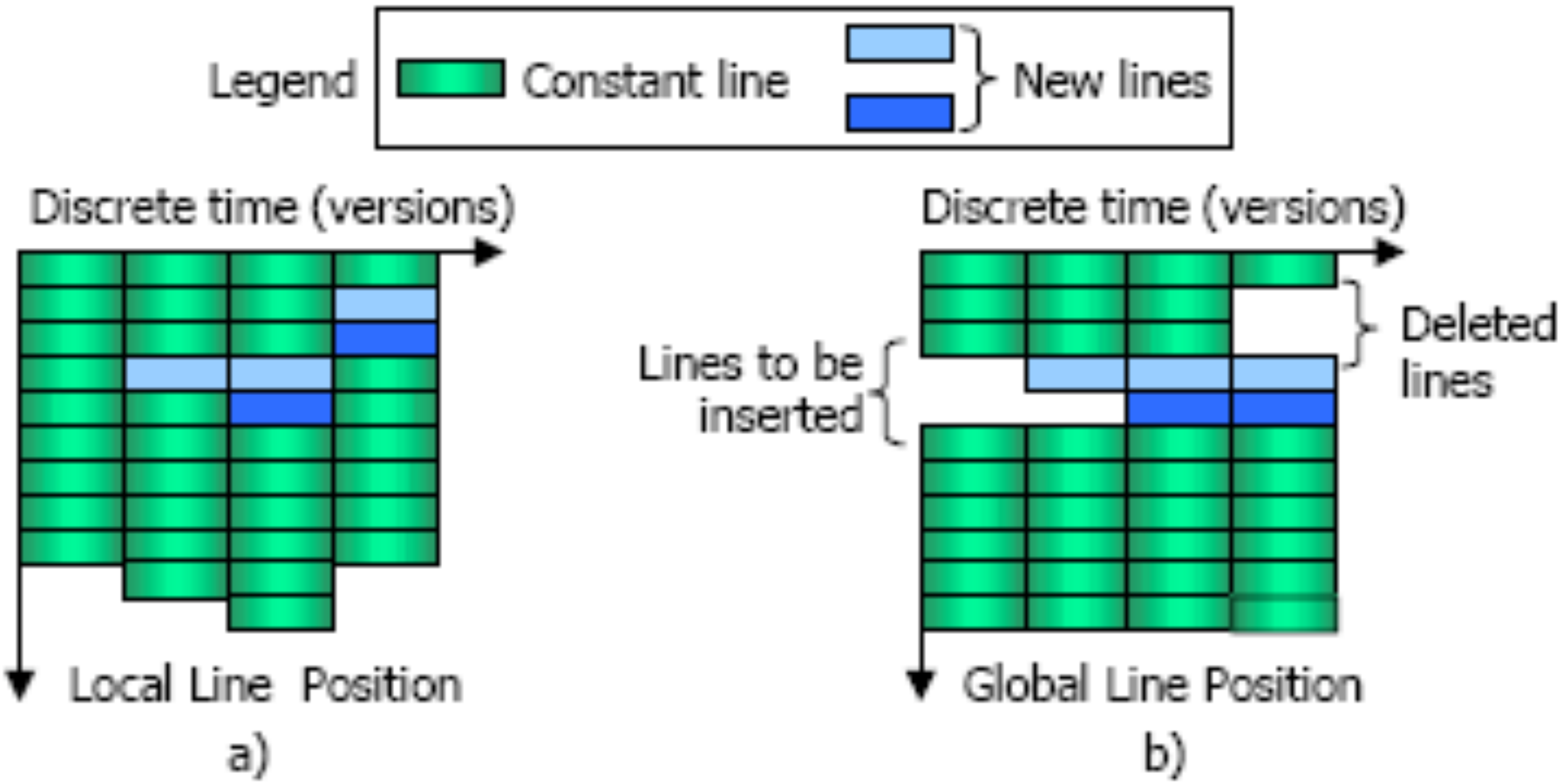


Figure 2 Global line position and corresponding graph analogy

Global Line Position (2)



Global Line Position allows Left to Right reading

Multiple Views

Figure 9: Multiple code views in CVSscan

2 Ways to Display Code

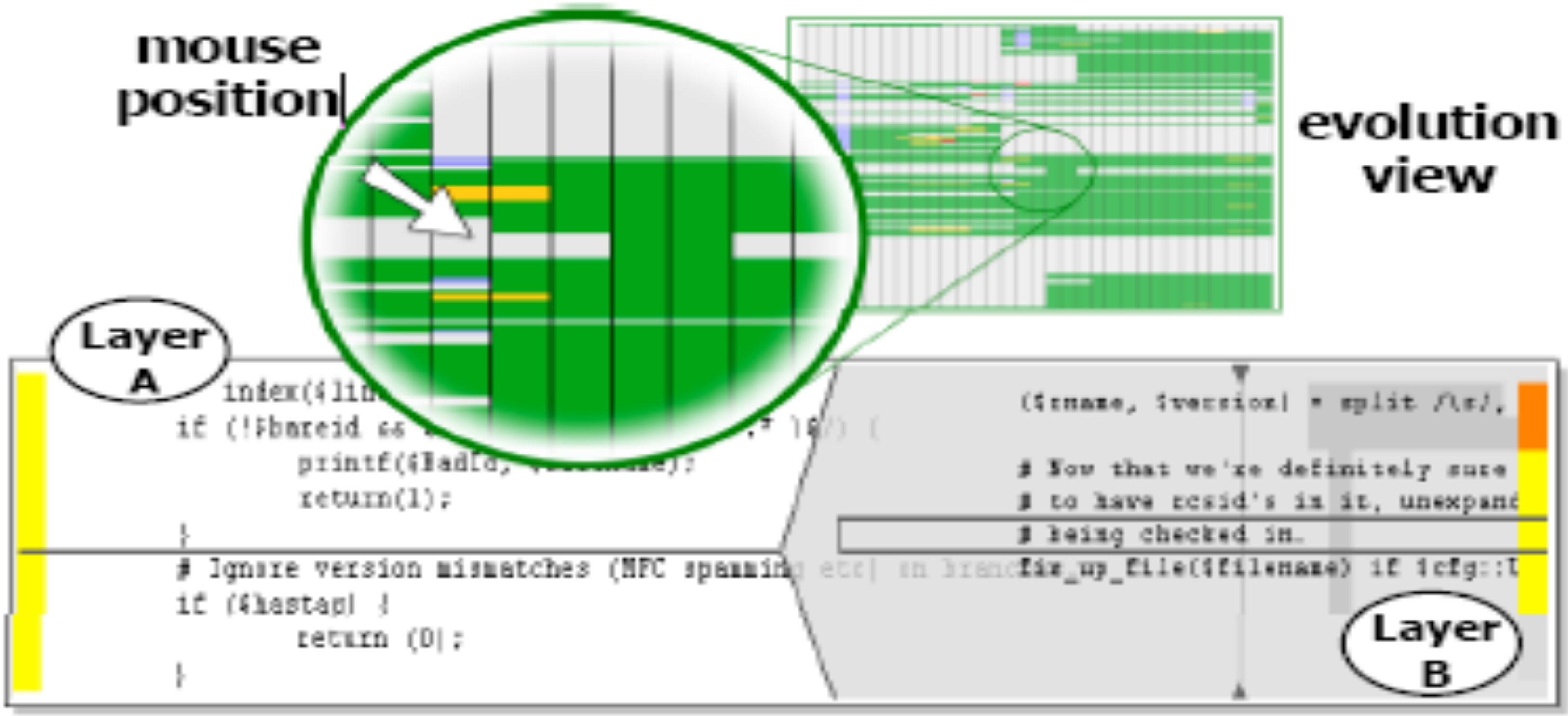


Figure 11: Two-layered code view

Use Case Validation

- Informal Studies (not targeted)
- 15 minutes of training
- Silent Observer
- Why not use a real-world case? (i.e. trying to fix a bug)
- No control
- No negative/constructive comments

Use Case #1

- Script file from the FreeBSD
- “Here they tuned the regular expressions”
- “Apparently a major change took place in the middle of the project. It mainly affected the check_version procedure”
- Rated as a success

Use Case #2

- C file socket implementation of the X Transport service layer
- The user recognized 2 authors performed most of the changes and the area of heavy modification
- Overall, the user did not have a very clear image of the file's evolution

Demo

Evaluation

- Who performed these modifications of the code?
  - (E) Hard to Track exactly “who is pink?”
- Which parts of the code are unstable?
  - (B) Seems o.k. for this purpose
- How are changes correlated?
  - (F) Correlation to other files in same check-in?
  - Correlation to other changes in the same file?

Evaluation

- How are the development tasks distributed?
  - (D) Although we can see distribution, precisely who wrote what is difficult to figure out
- What is the context in which a piece of code appeared?
  - (F) Hard to link back to changelist
  - Branching history?

Visualizing Application Behavior on Superscalar Processors

- Chris Stolte et al.

What is it?

- Program called Rivet
- Help optimization on multi-processor architectures

Motivation

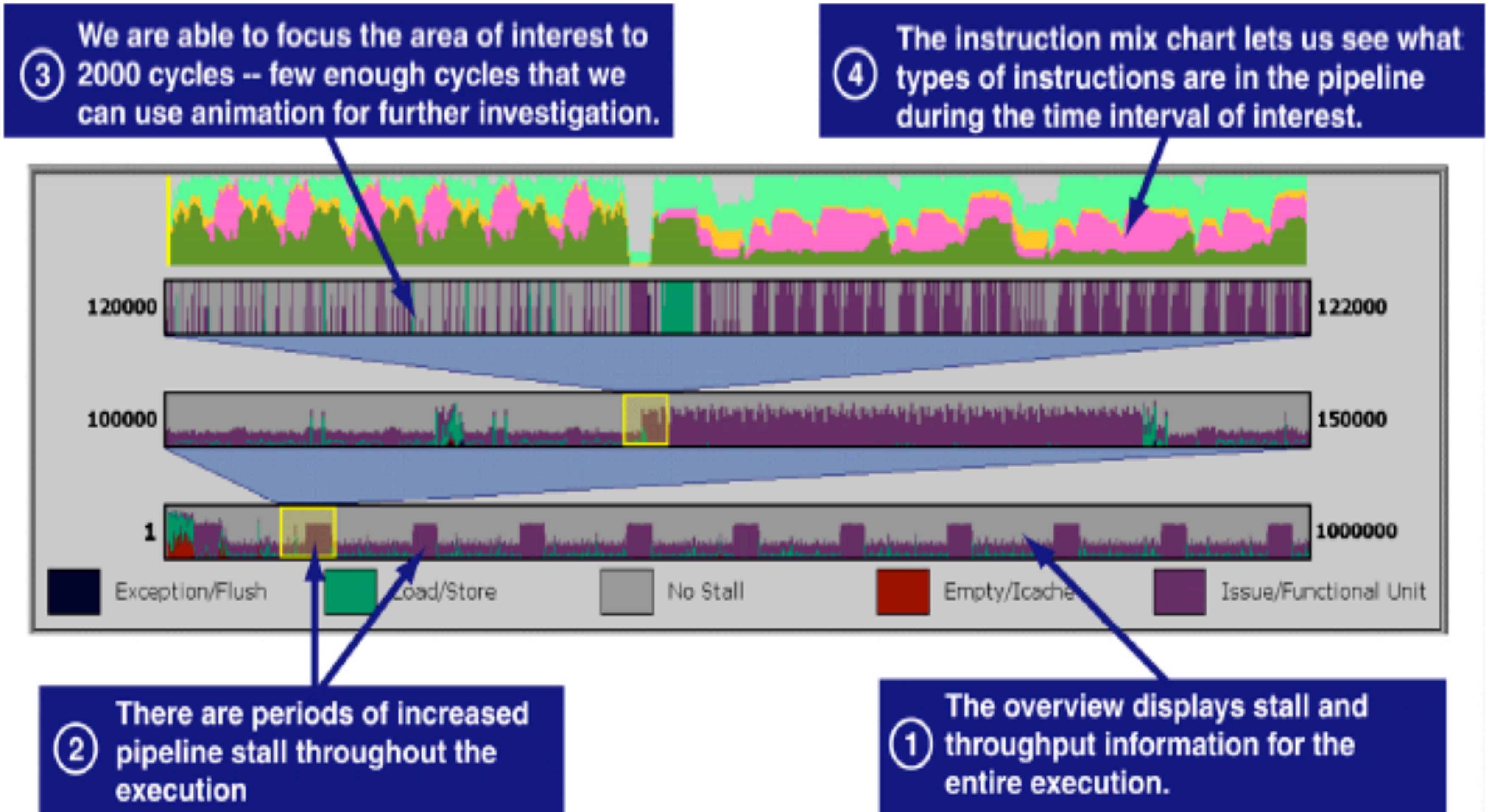
- Optimize
  - Know where to look
  - Drill into the details
  - Know the context – map back to the source code somehow

Main Optimization Techniques

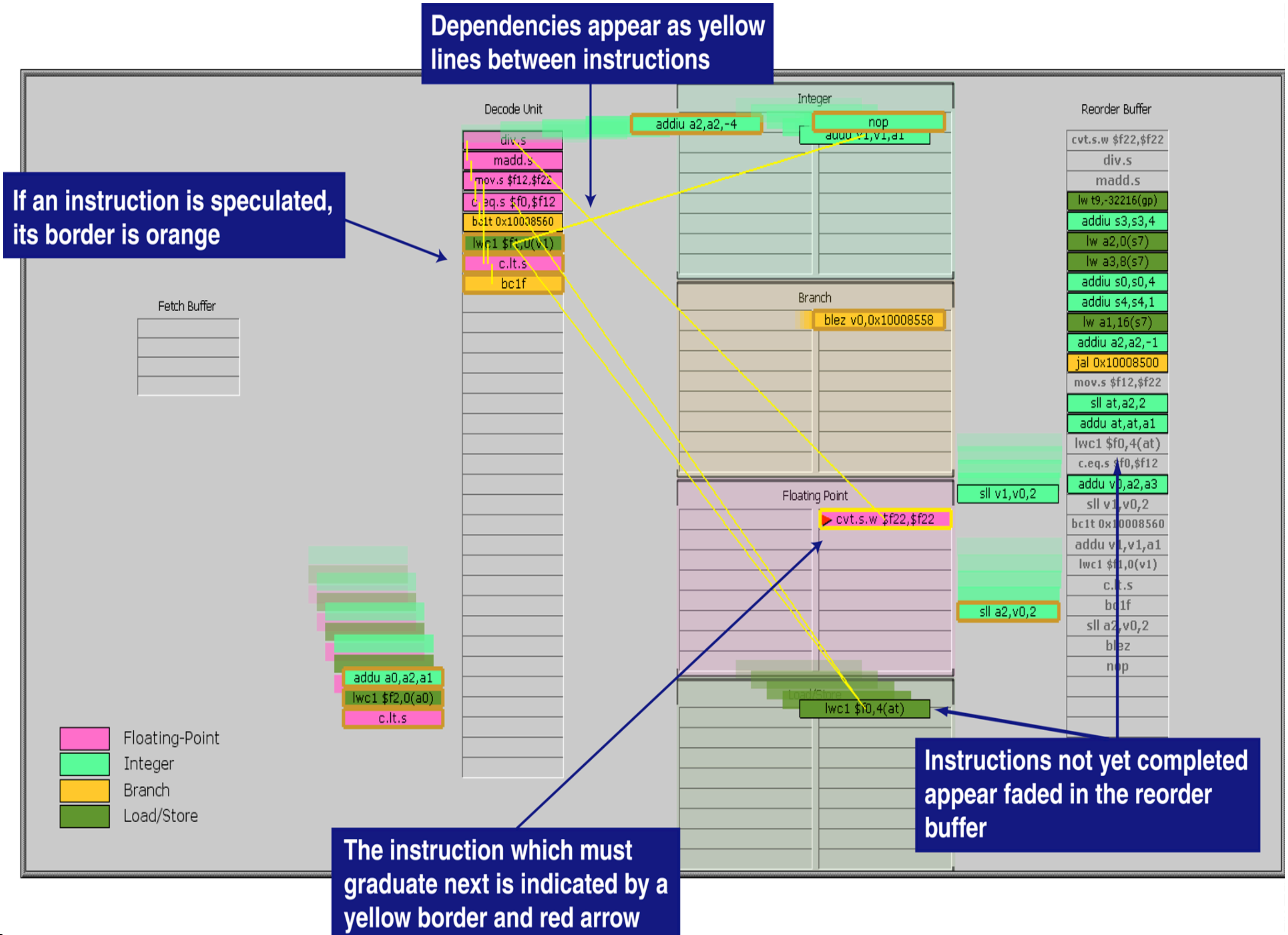
- *Pipelining:* overlap the execution of multiple instructions within a functional unit
- *Multiple Functional Units:* exploit instruction level parallelism (ILP)
- *Out-of-Order Execution:* increase possibility of ILP
- *Speculation:* guess and fetch ahead

What the program tracks

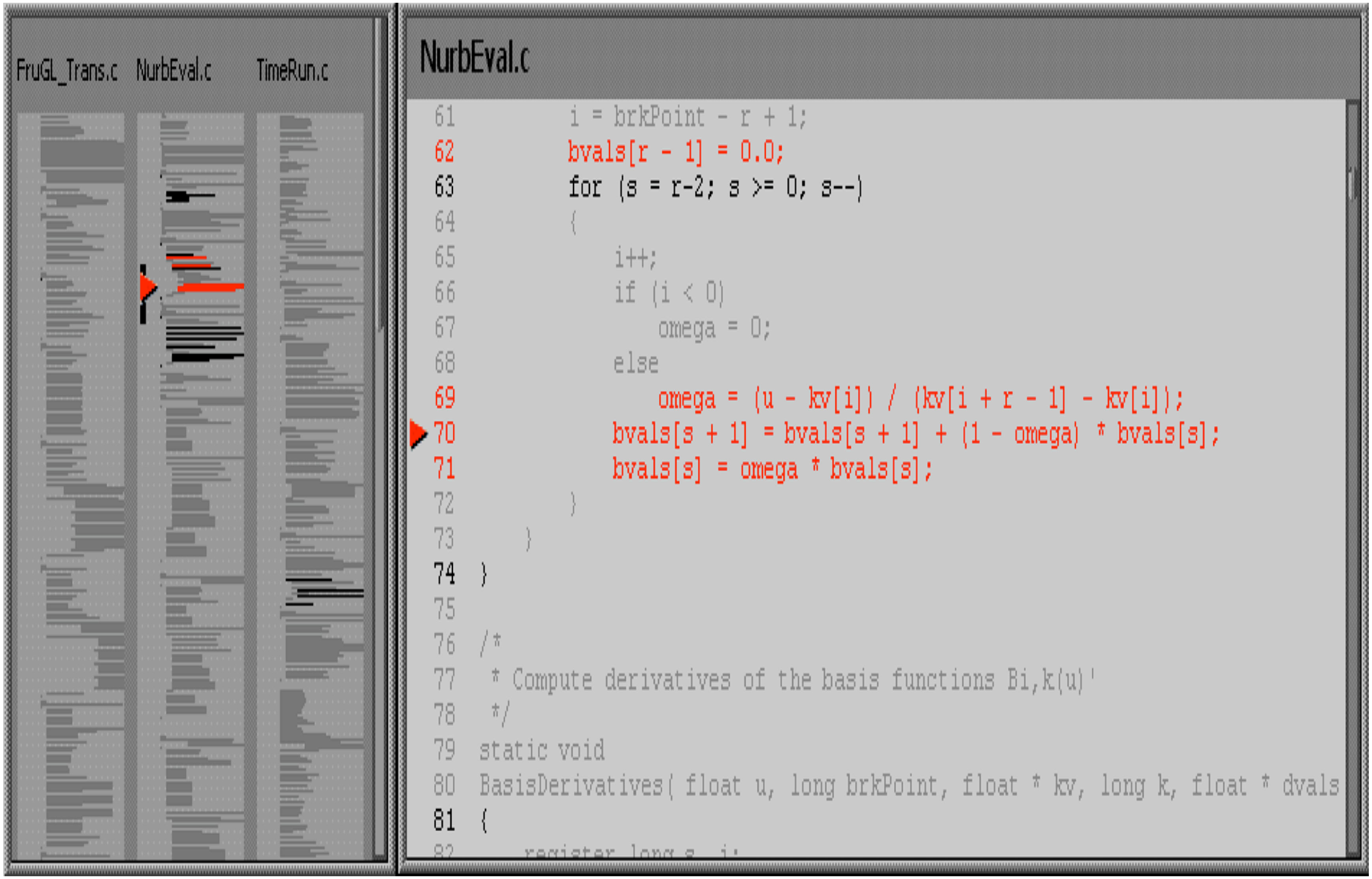
- *Empty/Icache:* An instruction cache miss
- *Exception/Flush :* An instruction requires sequential execution
- *Load/Store:* Waiting for memory
- *Issue/Functional Unit:* Waiting for a functional unit to complete execution

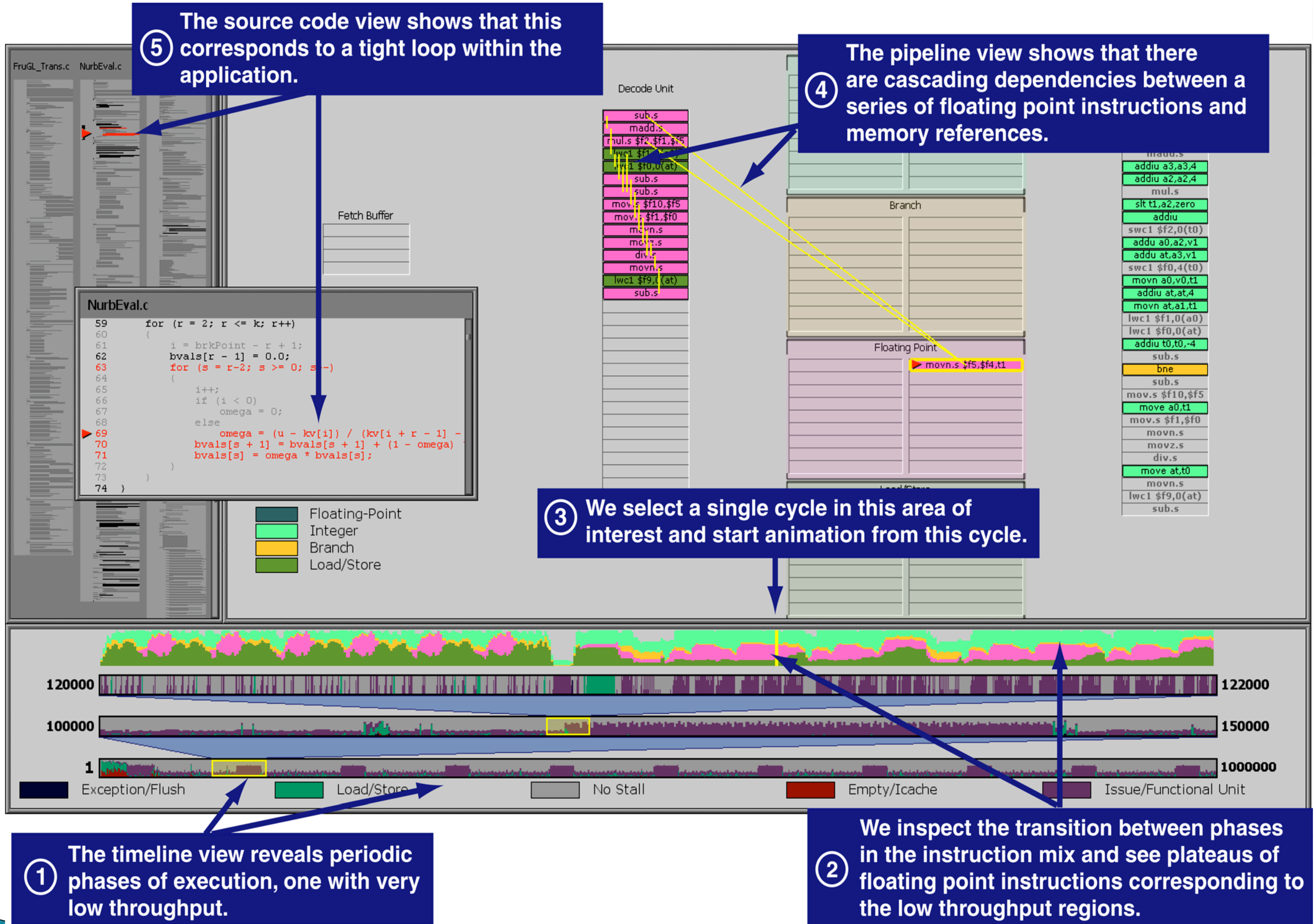


3 We are able to focus the area of interest to 2000 cycles -- few enough cycles that we can use animation for further investigation.
4 The instruction mix chart lets us see what types of instructions are in the pipeline during the time interval of interest.
120000
122000
100000
150000
1
1000000
Exception/Flush
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
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
Decode Unit
Fetch Buffer
Integer
Branch
Floating Point
Load/Store
Reorder Buffer
Floating-Point
Integer
Branch
Load/Store
div.s
madd.s
mov.s $f12,$f22
c.eq.s $f0,$f12
bc1t 0x10008560
c.lt.s
bc1f
addiu a2,a2,-4
nop
blez v0,0x10008558
cvt.s.w $f22,$f22
lwc1 $f0,4(at)
sll v1,v0,2
sll a2,v0,2
addu a0,a2,a1
lwc1 $f2,0(a0)
cvt.s.w $f22,$f22
lw t9,-32216(gp)
addiu s3,s3,4
lw a2,0(s7)
lw a3,8(s7)
addiu s0,s0,4
addiu s4,s4,1
lw a1,16(s7)
addiu a2,a2,-1
jal 0x10008500
mov.s $f12,$f22
sll at,a2,2
addu at,at,a1
lwc1 $f0,4(at)


FruGL_Trans.c
NurbEval.c
TimeRun.c
NurbEval.c
i = brkPoint - r + 1;
bvals[r - 1] = 0.0;
for (s = r-2; s >= 0; s--)
{
i++;
if (i < 0)
omega = 0;
else
omega = (u - kv[i]) / (kv[i + r - 1] - kv[i]);
bvals[s + 1] = bvals[s + 1] + (1 - omega) * bvals[s];
bvals[s] = omega * bvals[s];
}
}
}
/*
* Compute derivatives of the basis functions Bi,k(u)'
*/
static void
BasisDerivatives(float u, long brkPoint, float * kv, long k, float * dvals
{



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

Evaluation

- Know where to look.
  - (B) Great use of overview-plus detail display
  - But is this really the best entry point?
  - What about filters?
- Look at the details
  - (A) Looks good
- Know the context – map back to the source code somehow
  - (A) Looks good
  - Next step link to IDE?

Questions?