Phase-based Adaptive Recompilation in a JVM

Dayong Gu   Clark Verbrugge

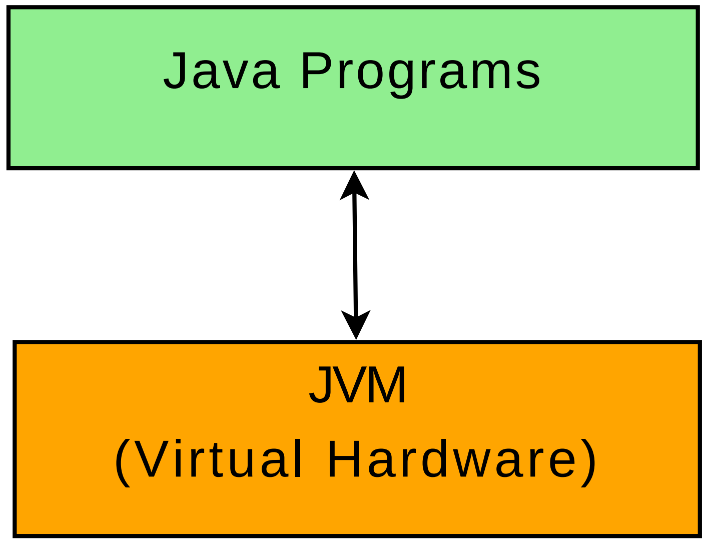
Sable Research Group, School of Computer Science
McGill University, Montréal, Canada
{dgu1, clump}@cs.mcgill.ca

April 7, 2008

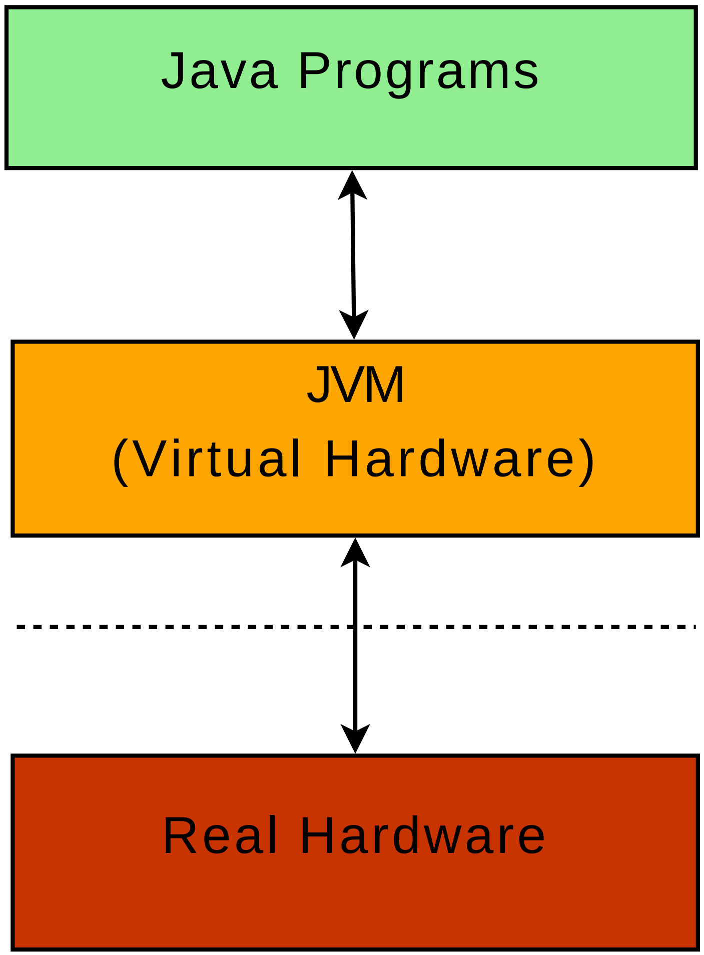
Outline

1. Motivation
2. Hardware Information Analysis
3. Adaptive Recompilation
4. Conclusions and Future Work

Motivation



Motivation



Motivation

Hardware Impact

- The impact of hardware on program behaviour can be significant
- Strong correlation exists: hardware performance $\leftrightarrow$ program behaviour

Hardware Event Counters

- Hardware counters widely exist in modern processors
- Accessible from software libs: **PAPI**, PMAPI,PCL,...

Motivation

Hardware Impact

- The impact of hardware on program behaviour can be significant
- Strong correlation exists: hardware performance $\leftrightarrow$ program behaviour

Hardware Event Counters

- Hardware counters widely exist in modern processors
- Accessible from software libs: **PAPI**, PMAPI,PCL,...

Use hardware event data to improve adaptive optimizations in JVM

Program Phases

There are different types of program phases

Program Phases

There are different types of program phases

Flat Phases

refer to the contiguous intervals where show a stable, flat performance on a type of sampled, profiling data

Periodic Phases

long term repetitions in execution, showing unstable performance but sharing similar patterns

Program Phases

There are different types of program phases

Flat Phases

refer to the contiguous intervals where show a stable, flat performance on a type of sampled, profiling data

Periodic Phases

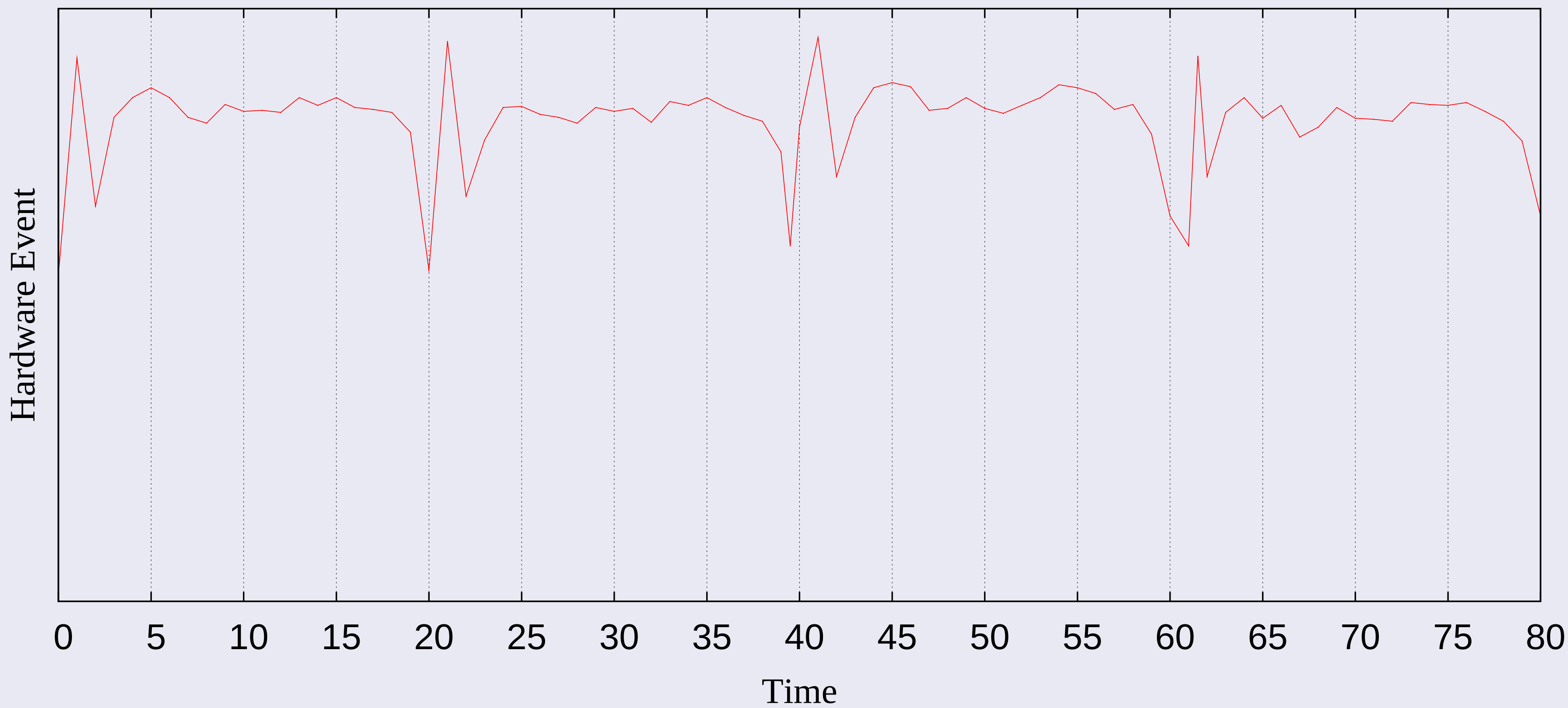
long term repetitions in execution, showing unstable performance but sharing similar patterns

Detect this long term **periodic phases** from hardware data and employ the phase information in adaptive optimizations

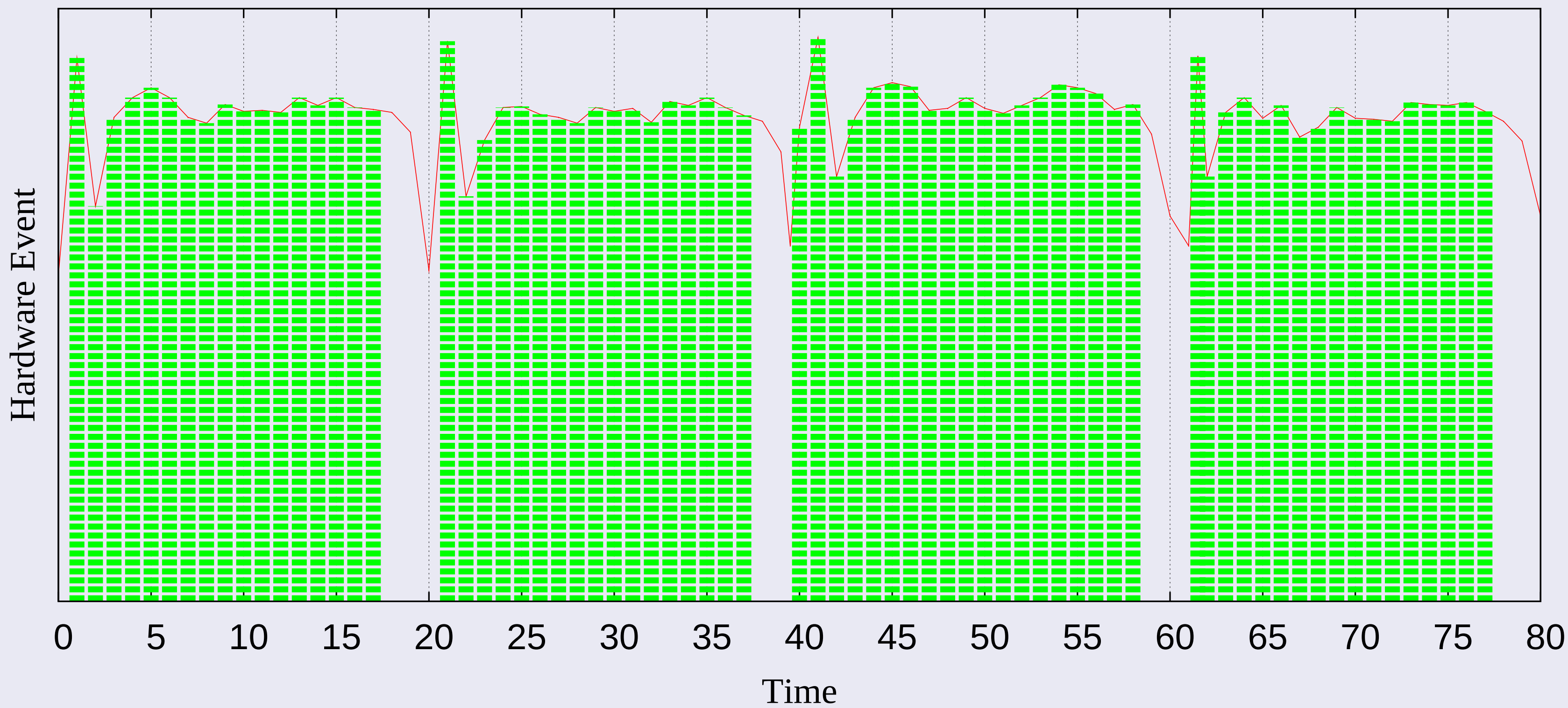
Contributions

- Highlight the hardware impact on Java program execution
- Develop online **pattern** creation algorithm to represent hardware event
- Detect long term periodic **phases** from hardware patterns
- Implement an **adaptive recomplication strategy** using phase information

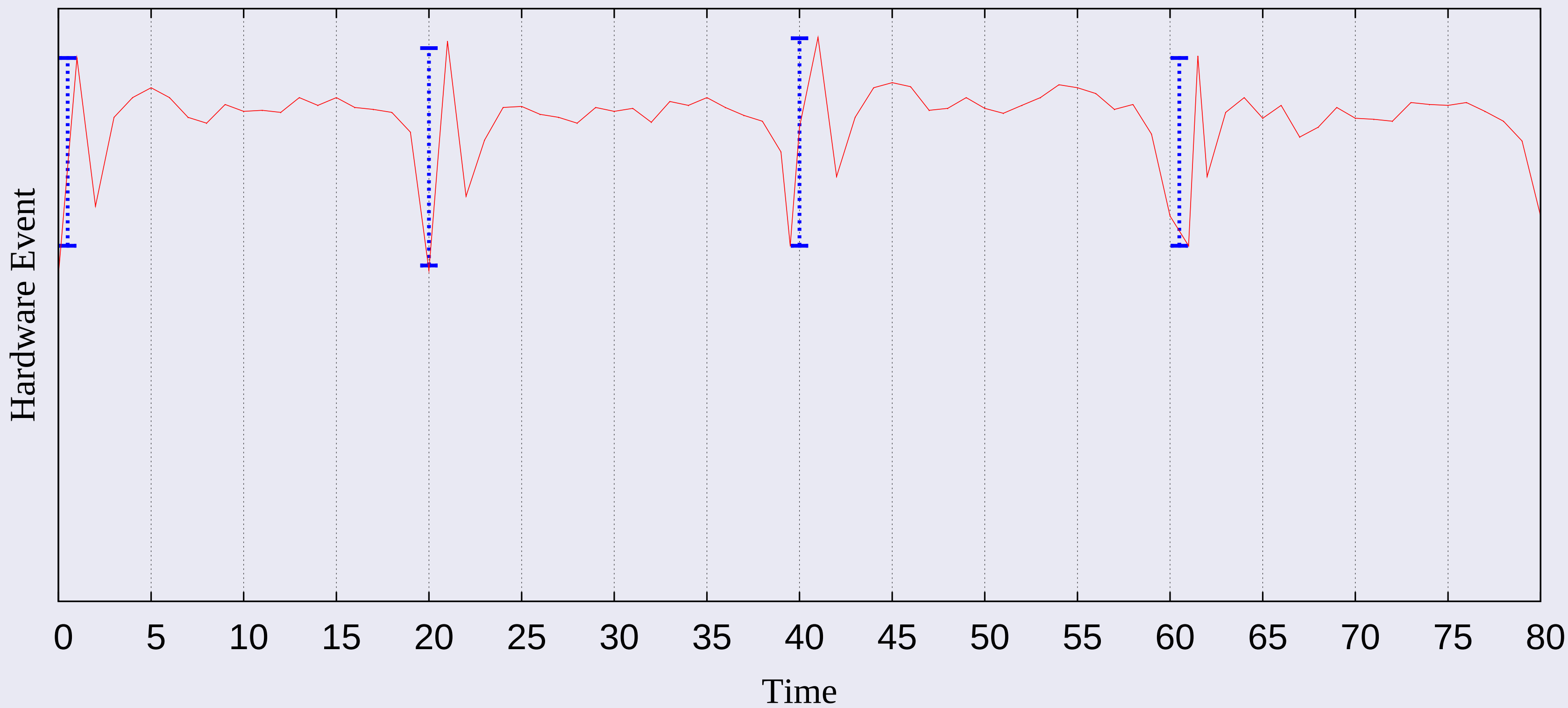
Hardware Event Data Example



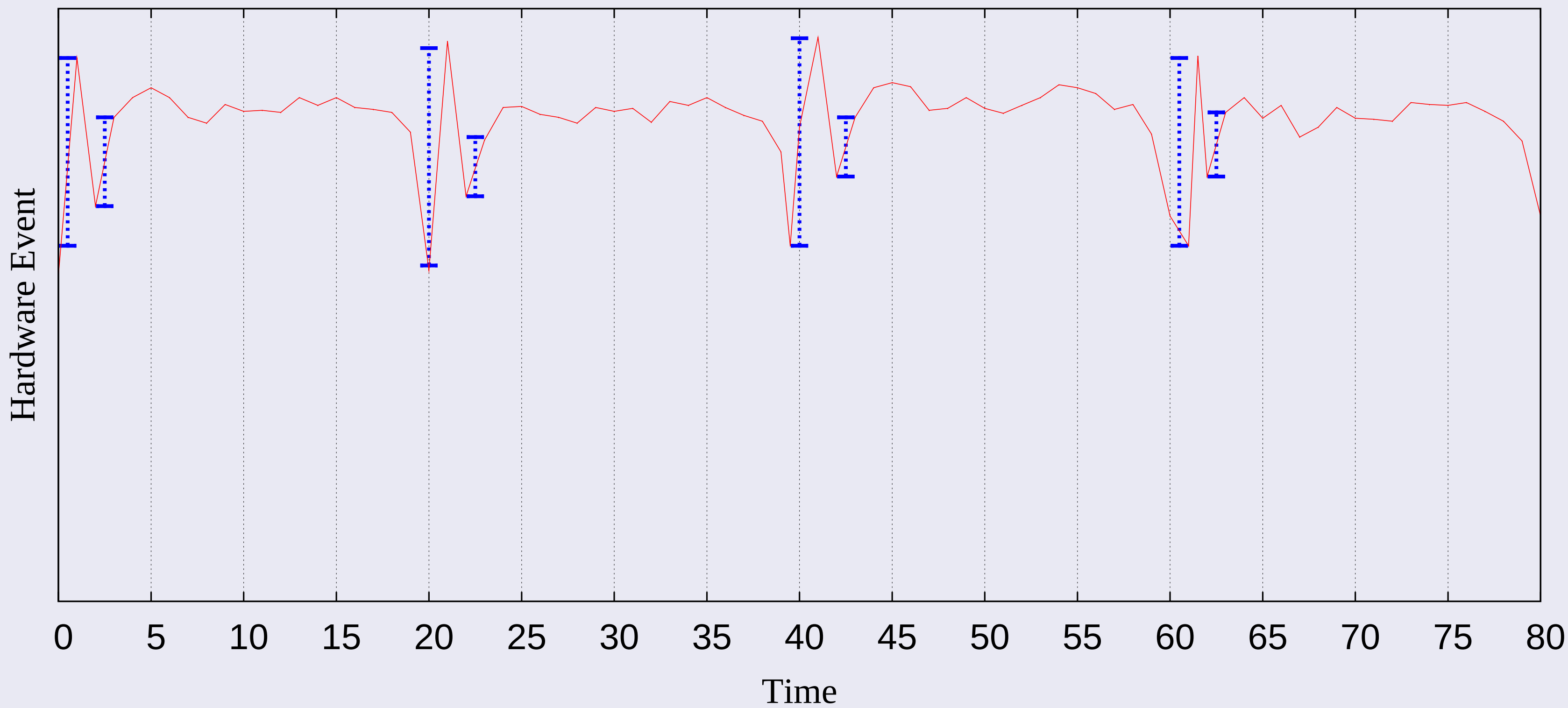
Recurrent Phases



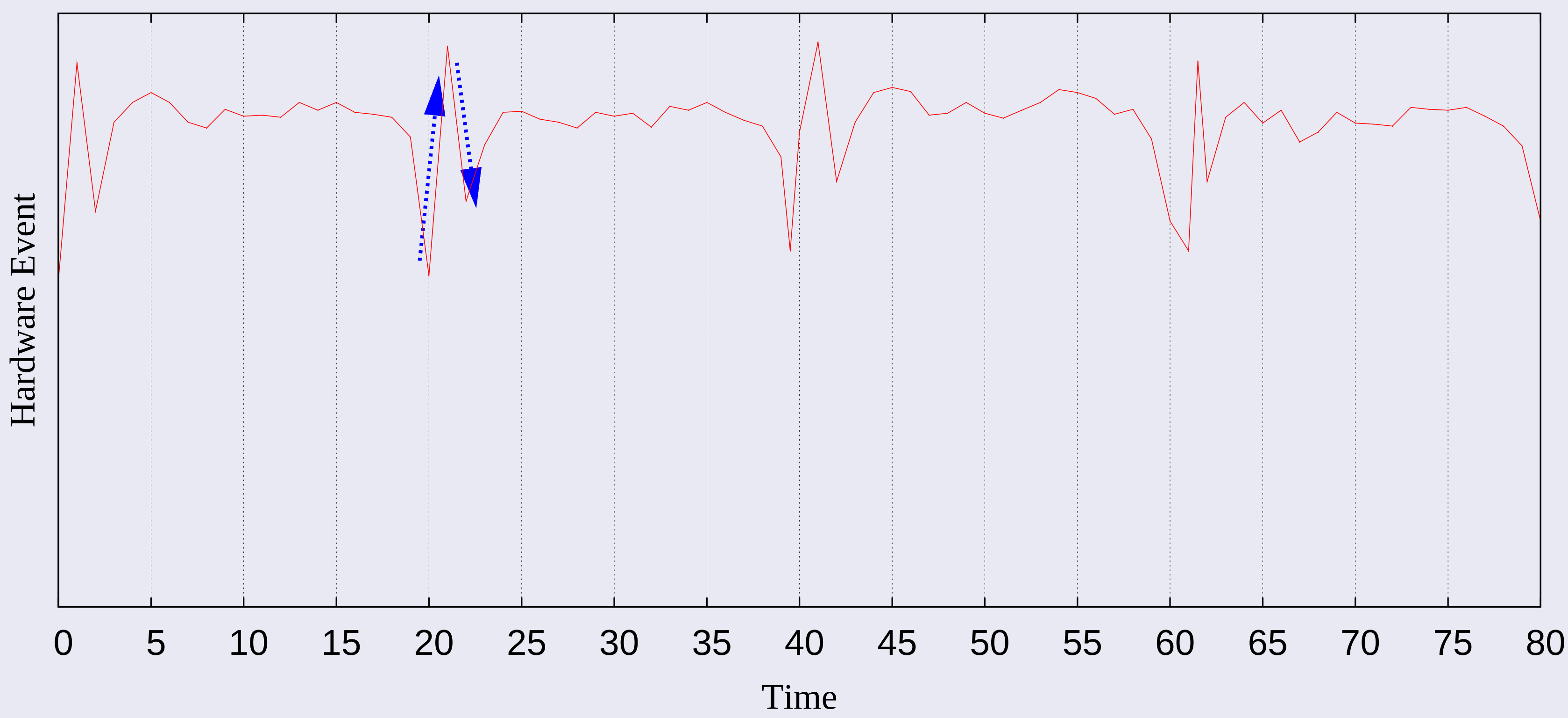
Hardware Pattern



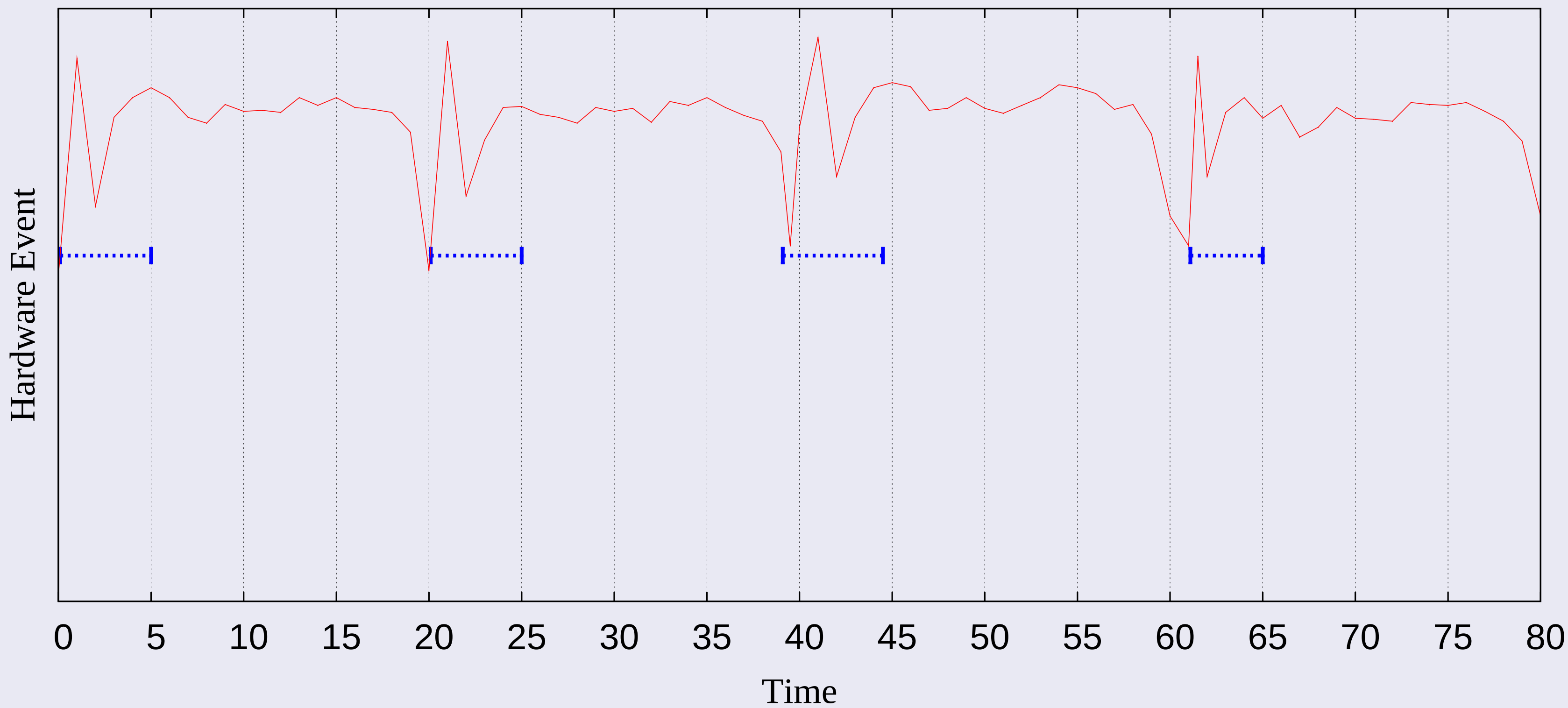
Hardware Pattern



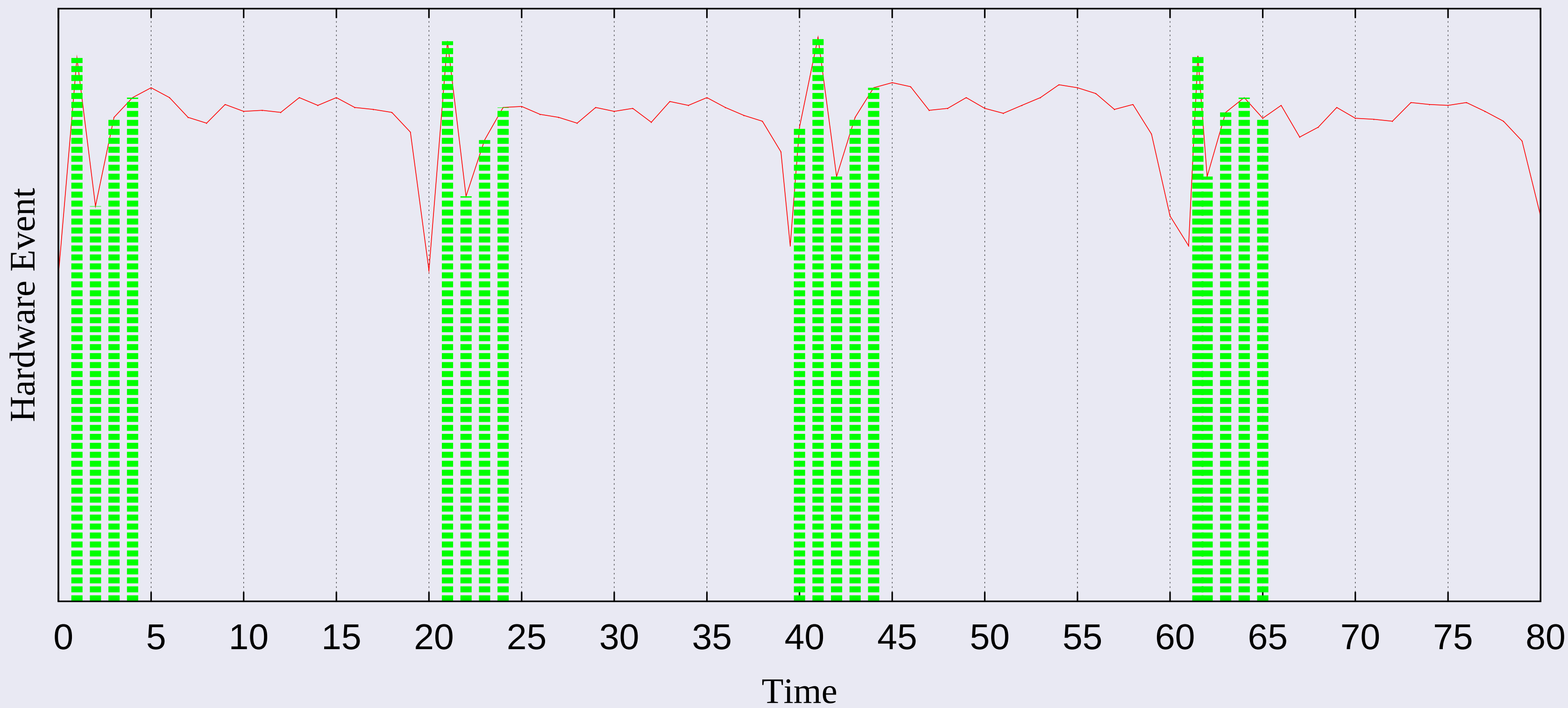
Hardware Pattern



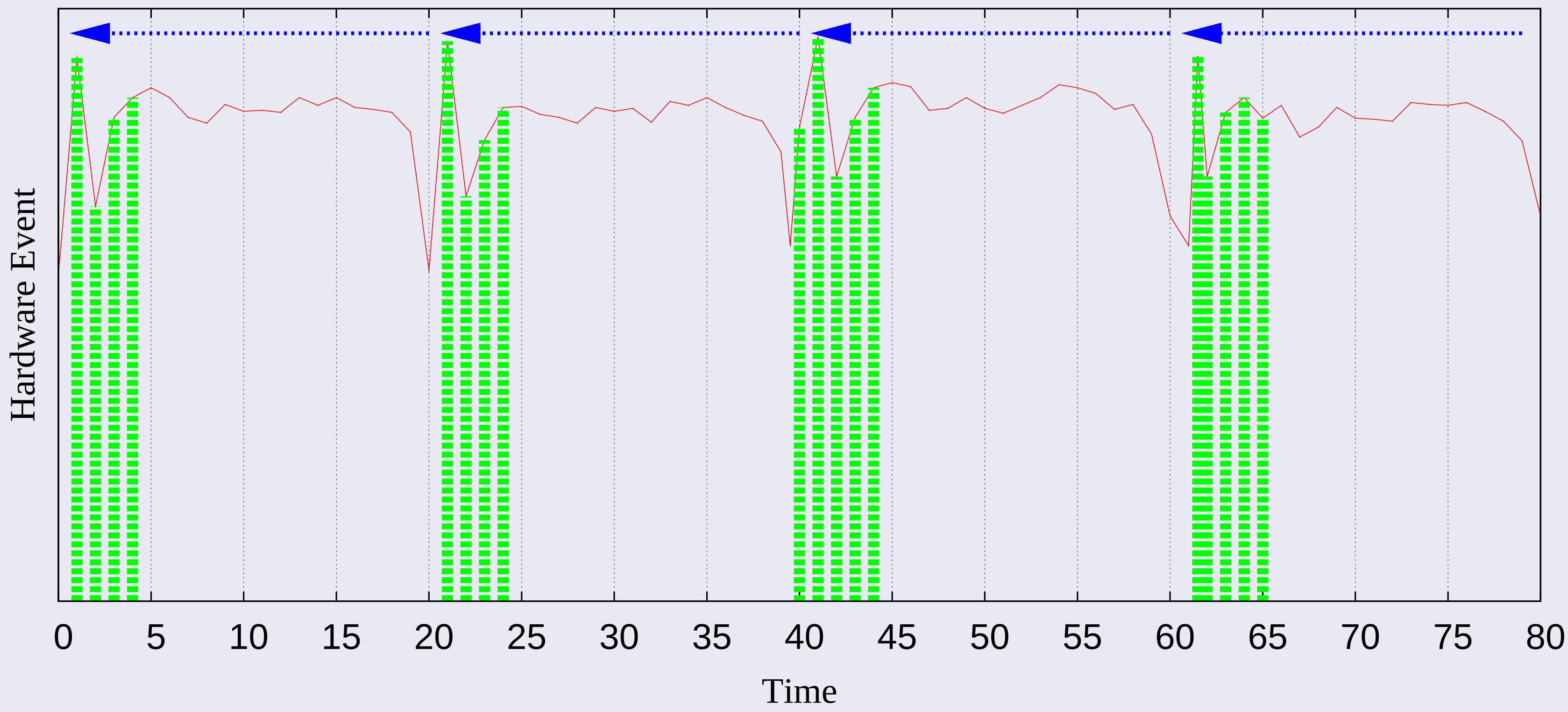
Hardware Pattern



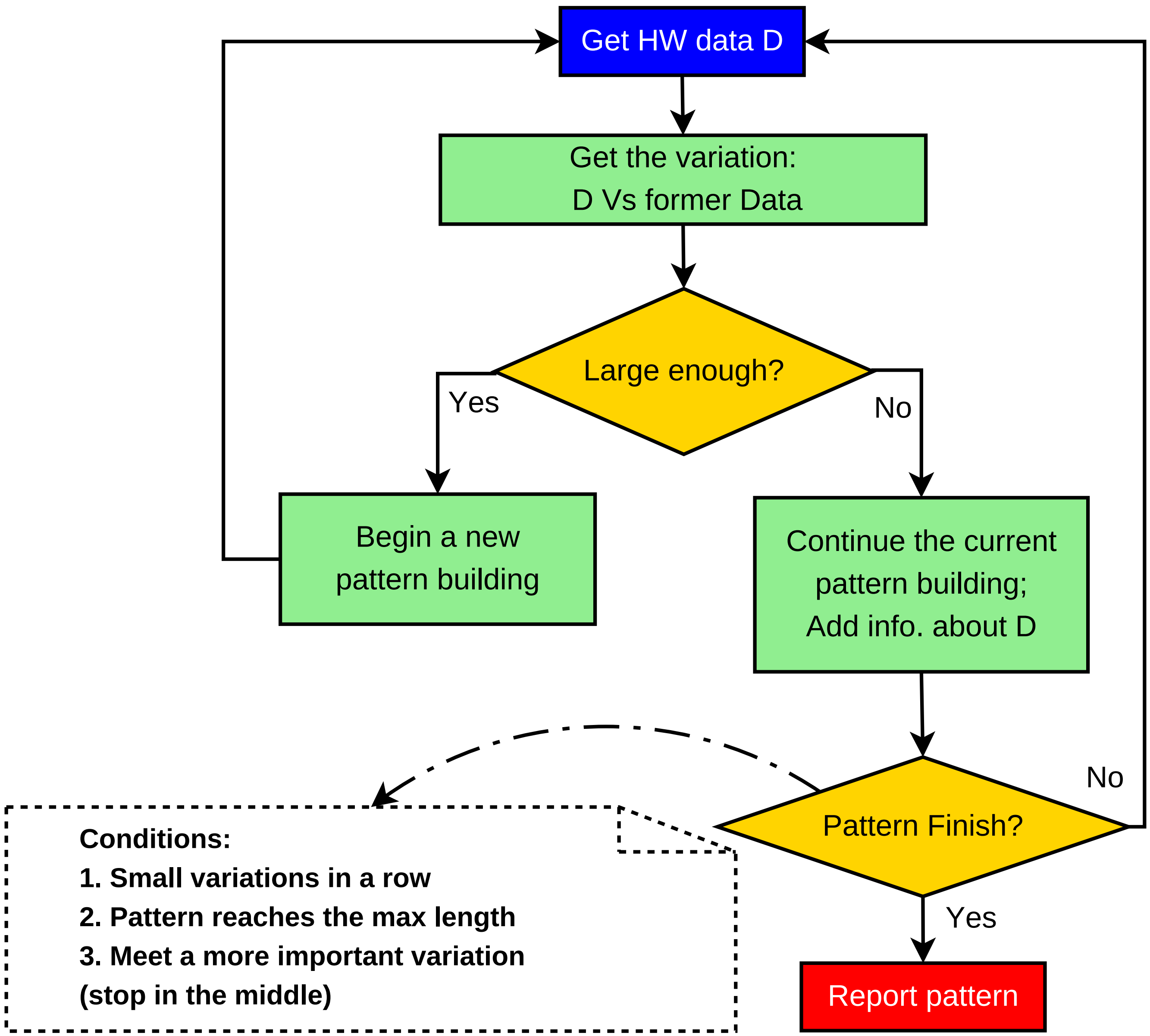
Hardware Pattern



Hardware Pattern

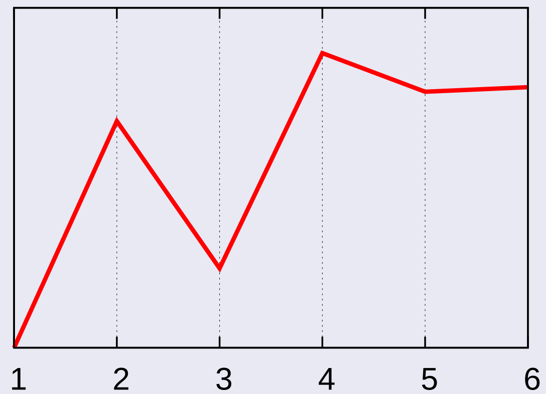


Pattern Building Algorithm (Simplified)



Pattern Building Example

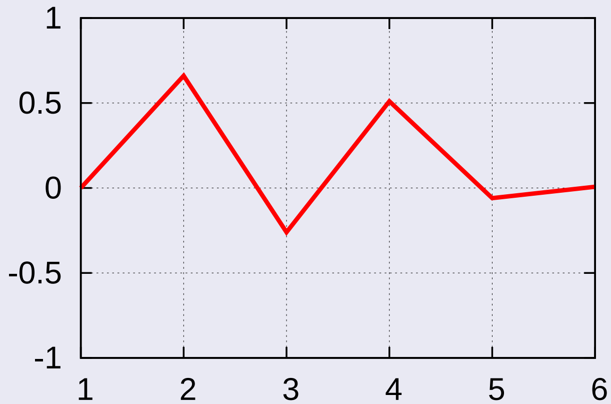
(1) Hardware Data



Pattern Building Example

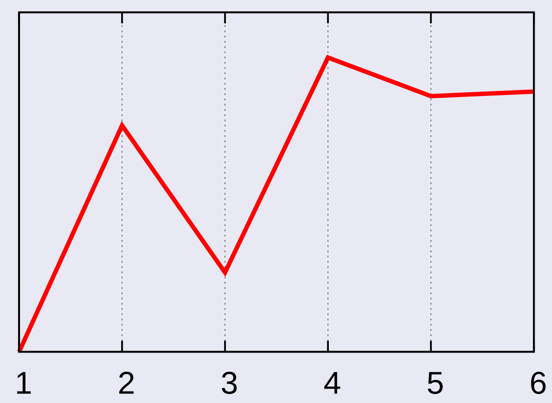
(1) Hardware Data

(2) Variation

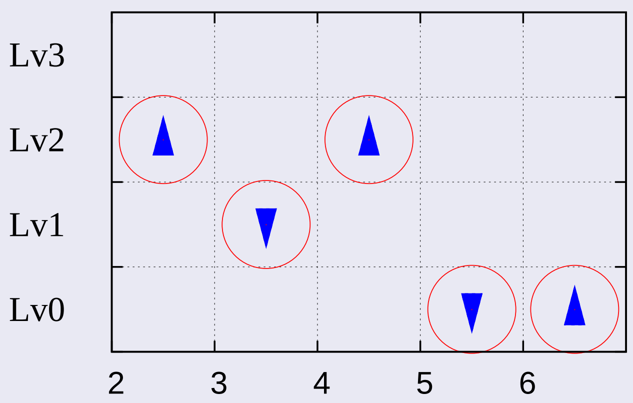


Pattern Building Example

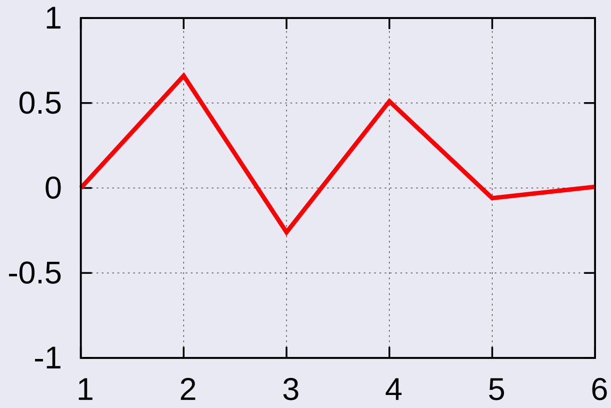
(1) Hardware Data



(3) Level

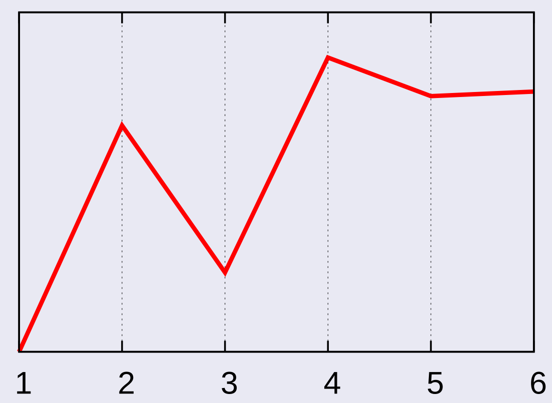


(2) Variation

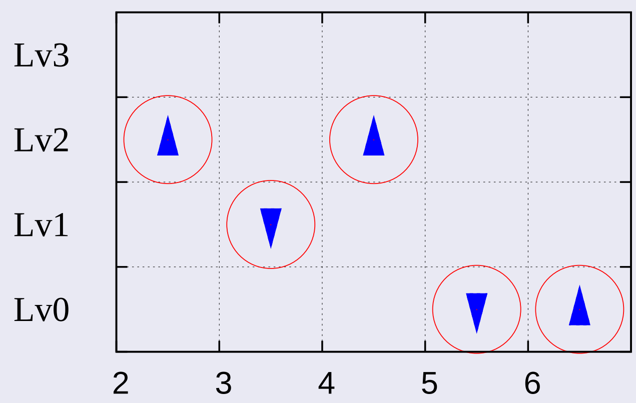


Pattern Building Example

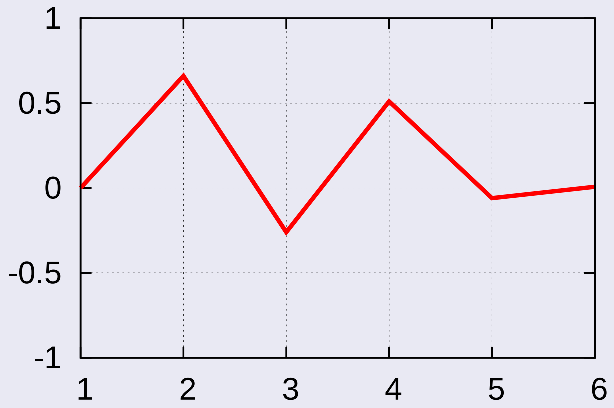
(1) Hardware Data



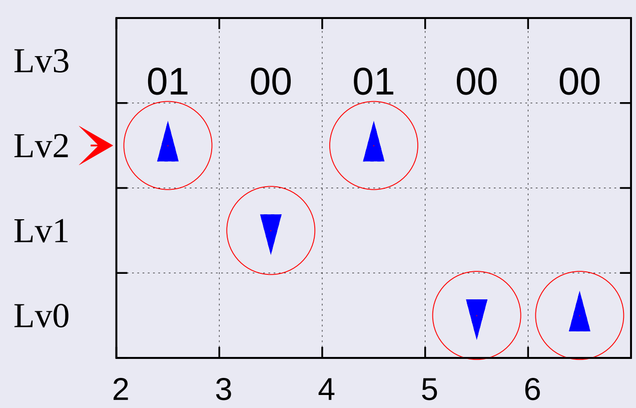
(3) Level



(2) Variation



(4) Pattern

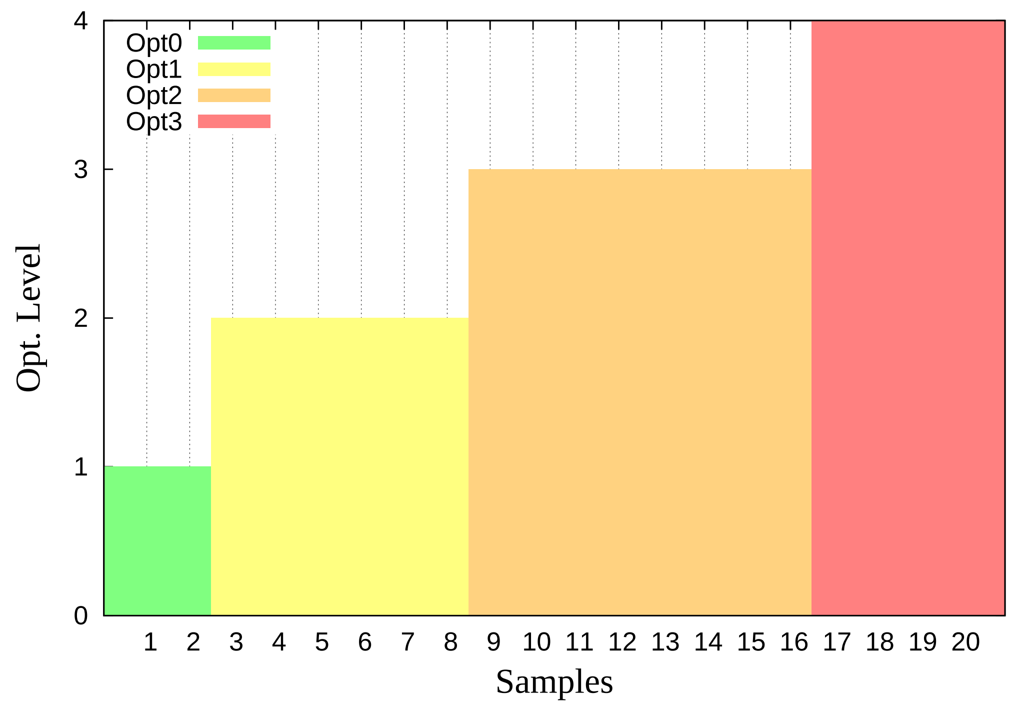


Pattern Analysis

- Patterns are stored and analyzed
- The number of occurrences determines the **hotness** of a pattern
- The hottest pattern is used to represent the current program **phase**
- The **phase information** is used to archieve a better adaptive hot method recompilation strategy in Jikes RVM

Recompilation in Jikes RVM

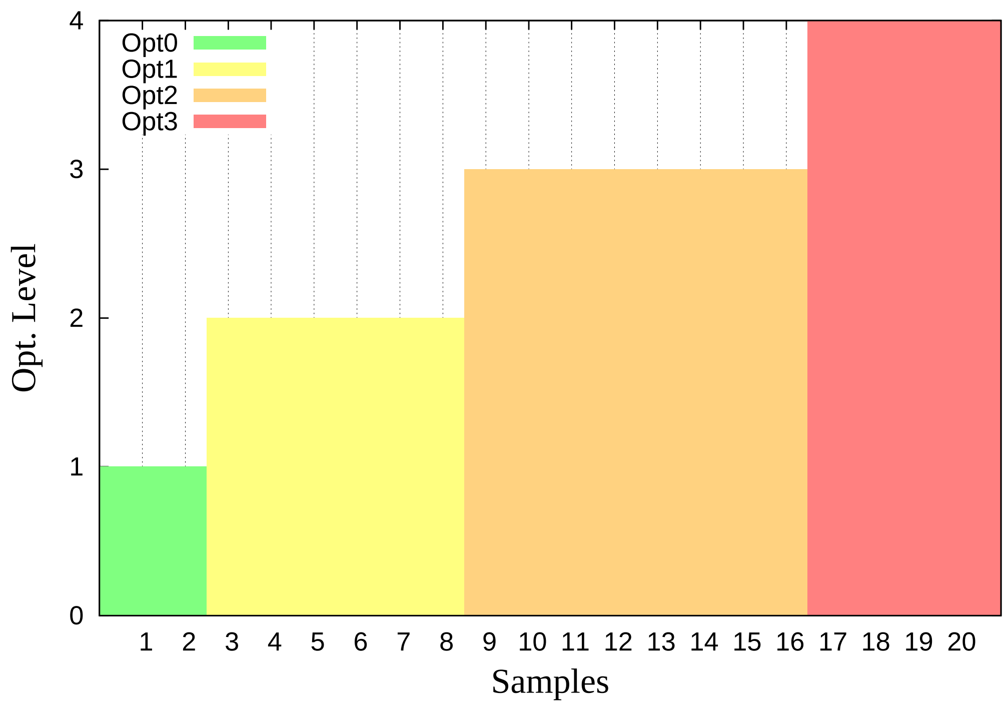
- Get method samples
- Compute the *Past* time in a method
- Estimate the compilation cost C_i to optimization level i



Recompile to level i, if $(SpeedupRate_i * Past) > C_i$

Recompilation in Jikes RVM

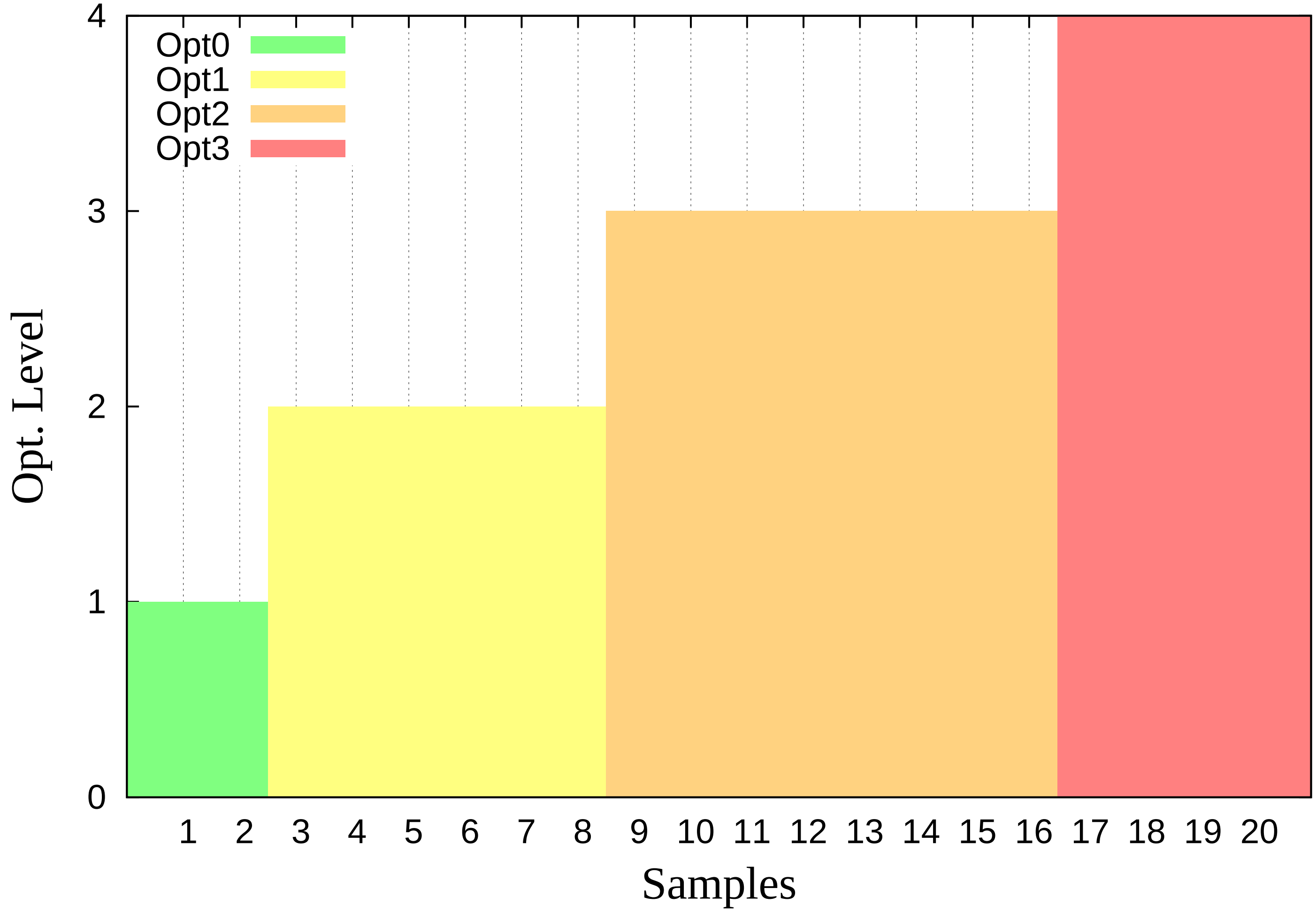
- Get method samples
- Compute the *Past* time in a method
- Estimate the compilation cost C_i to optimization level i



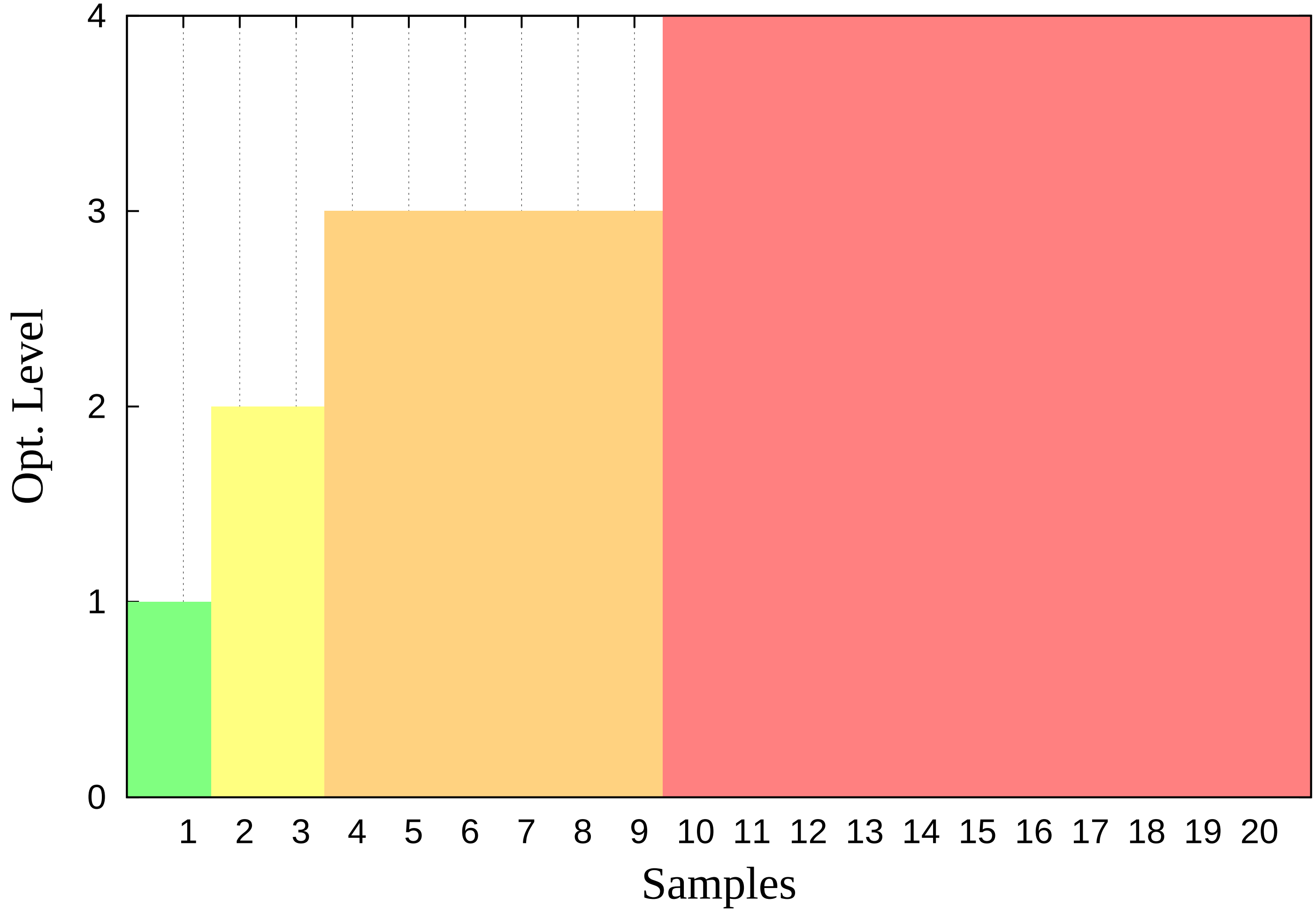
Recompile to level i, if $(SpeedupRate_i * Past) > C_i$

Assume $Future \equiv Past$

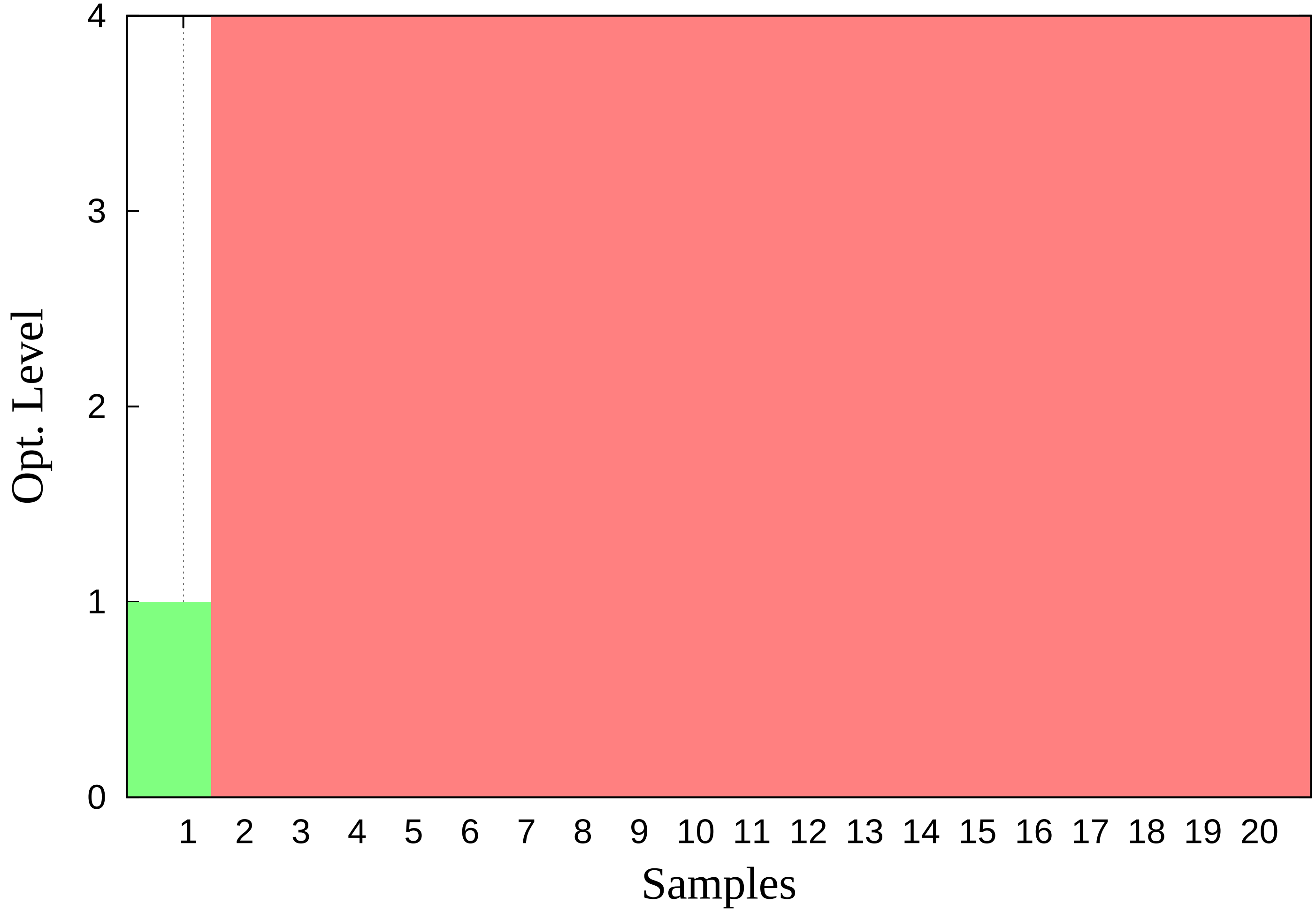
Optimization Opportunity



Optimization Opportunity



Optimization Opportunity



Optimization Opportunity

- Recompilation is not free, cannot always be aggressive

Optimization Opportunity

- Recompilation is not free, cannot always be aggressive
- Our approach:

Program State	Sampled Methods	Hardware Event Behaviour	Recompilation Aggressiveness
New	The "Beginners"	No recurrence of patterns	Low
Young	Important methods	Recurrences of patterns	High
Mature	Optimized methods Less important methods	Less fresh patterns More old patterns	Moderate
Rejuvenated	New important methods	More fresh patterns again	High

Optimization Opportunity

- Recompilation is not free, cannot always be aggressive
- Our approach:

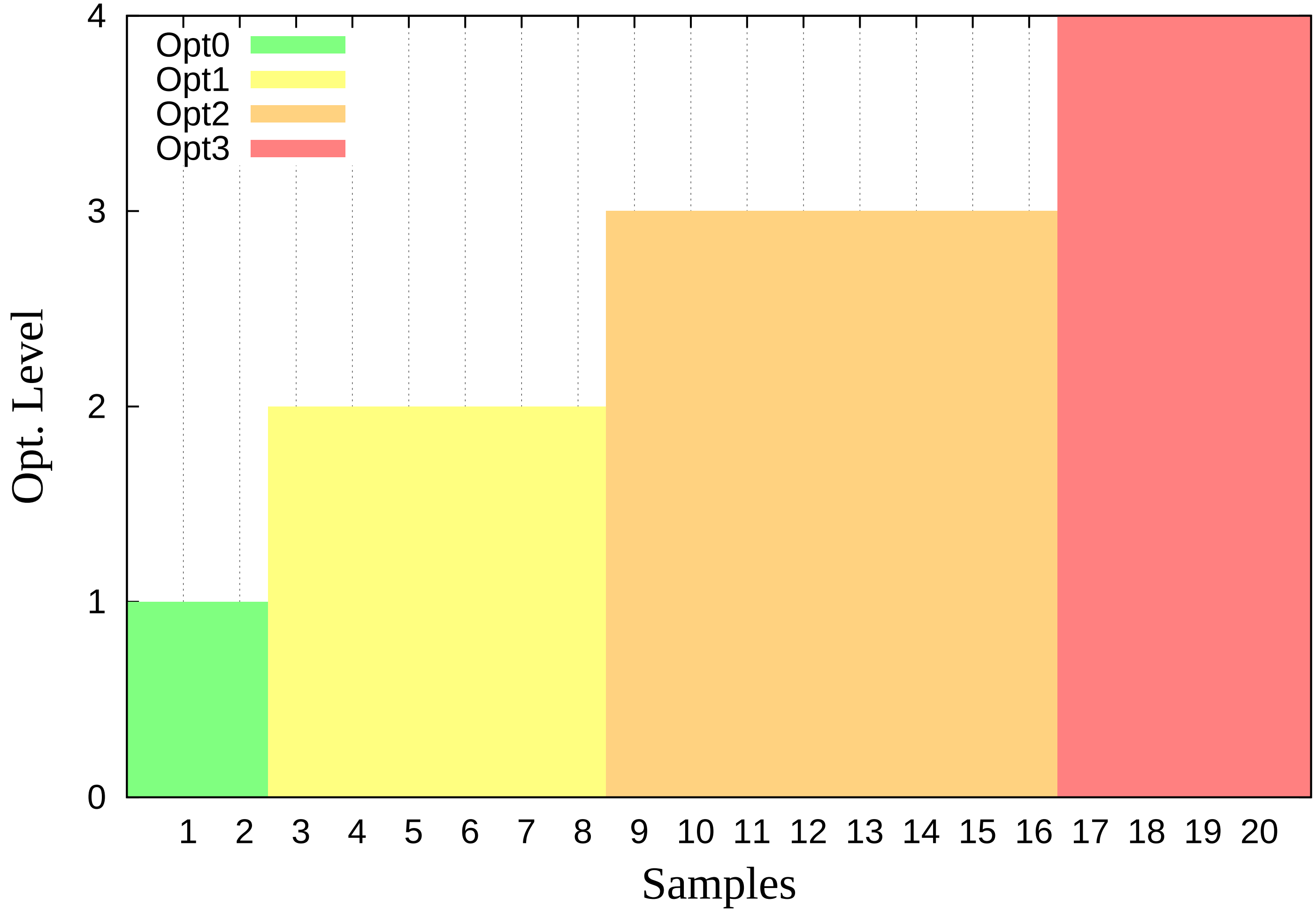
Program State	Sampled Methods	Hardware Event Behaviour	Recompilation Aggressiveness
New	The "Beginners"	No recurrence of patterns	Low
Young	Important methods	Recurrences of patterns	High
Mature	Optimized methods Less important methods	Less fresh patterns More old patterns	Moderate
Rejuvenated	New important methods	More fresh patterns again	High

- *Future $\neq$ Past*
    - Fixed aggressiveness $\Rightarrow$ Adaptive aggressiveness

Optimization Opportunity

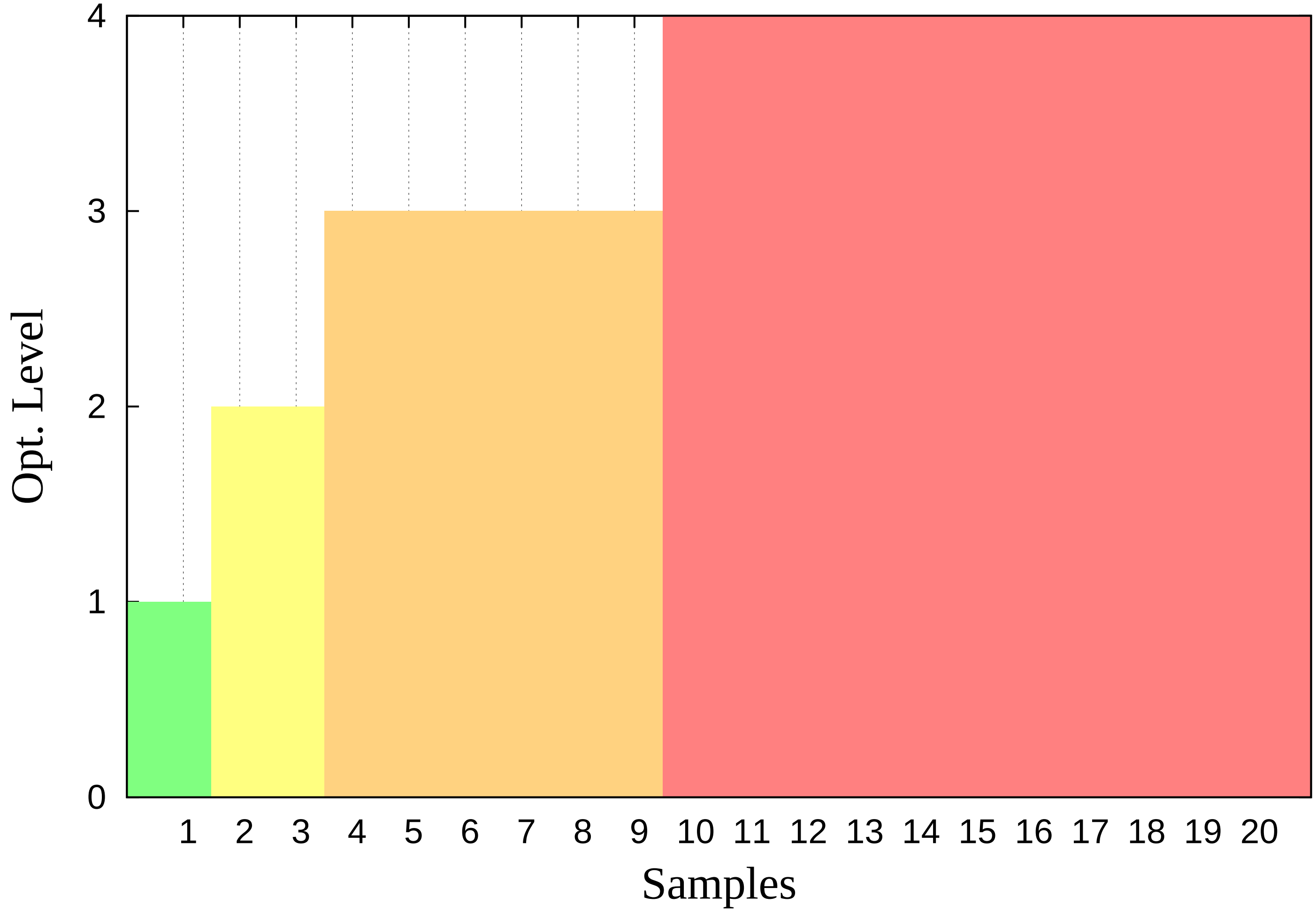
- Optimize code to higher levels earlier
- Possibly save recompilation overhead for intermediate levels
- Save unnecessary recompilation for the "beginners"

Original

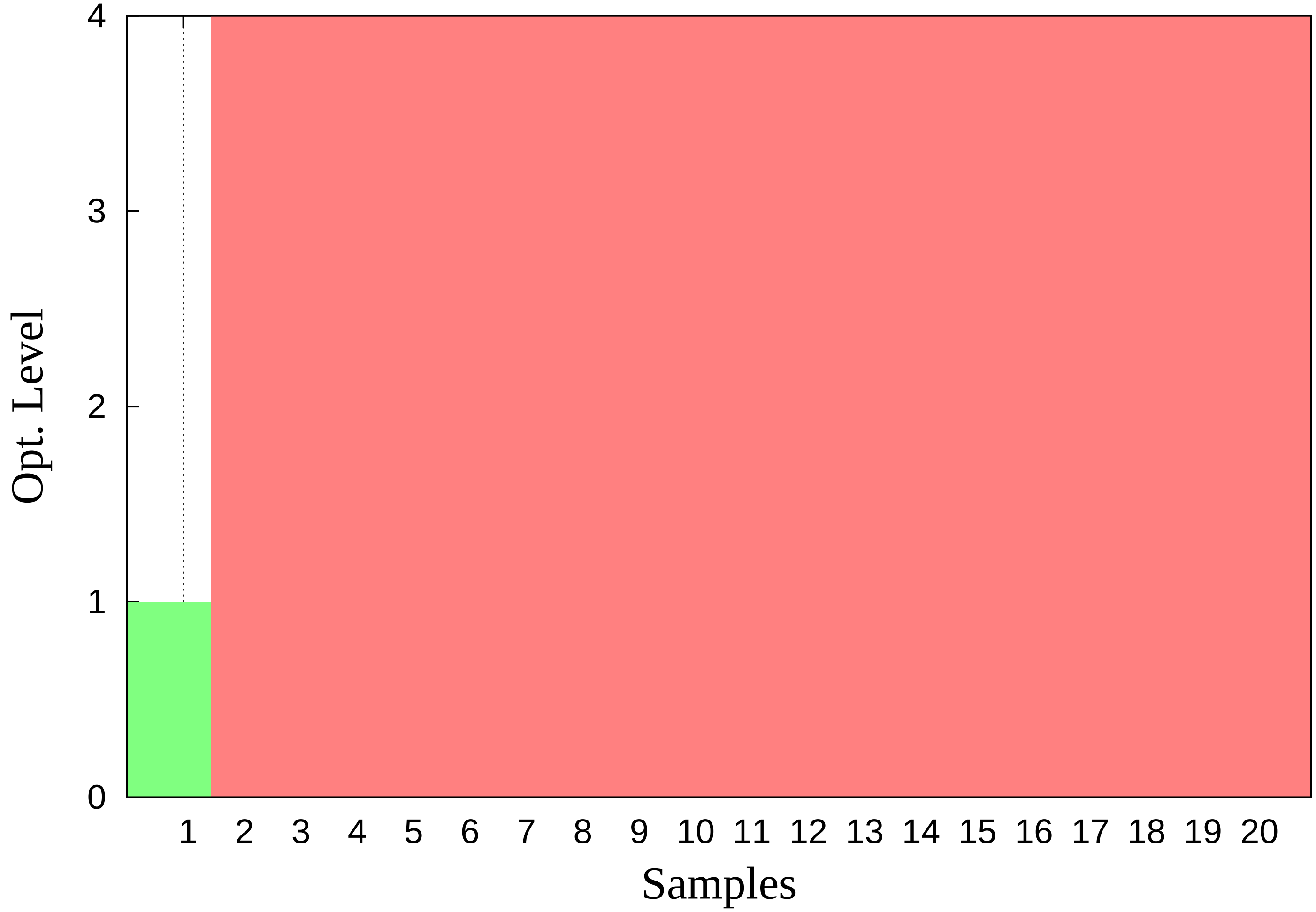


Opt0
Opt1
Opt2
Opt3
Opt. Level
Samples
0
1
2
3
4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Online Optimization



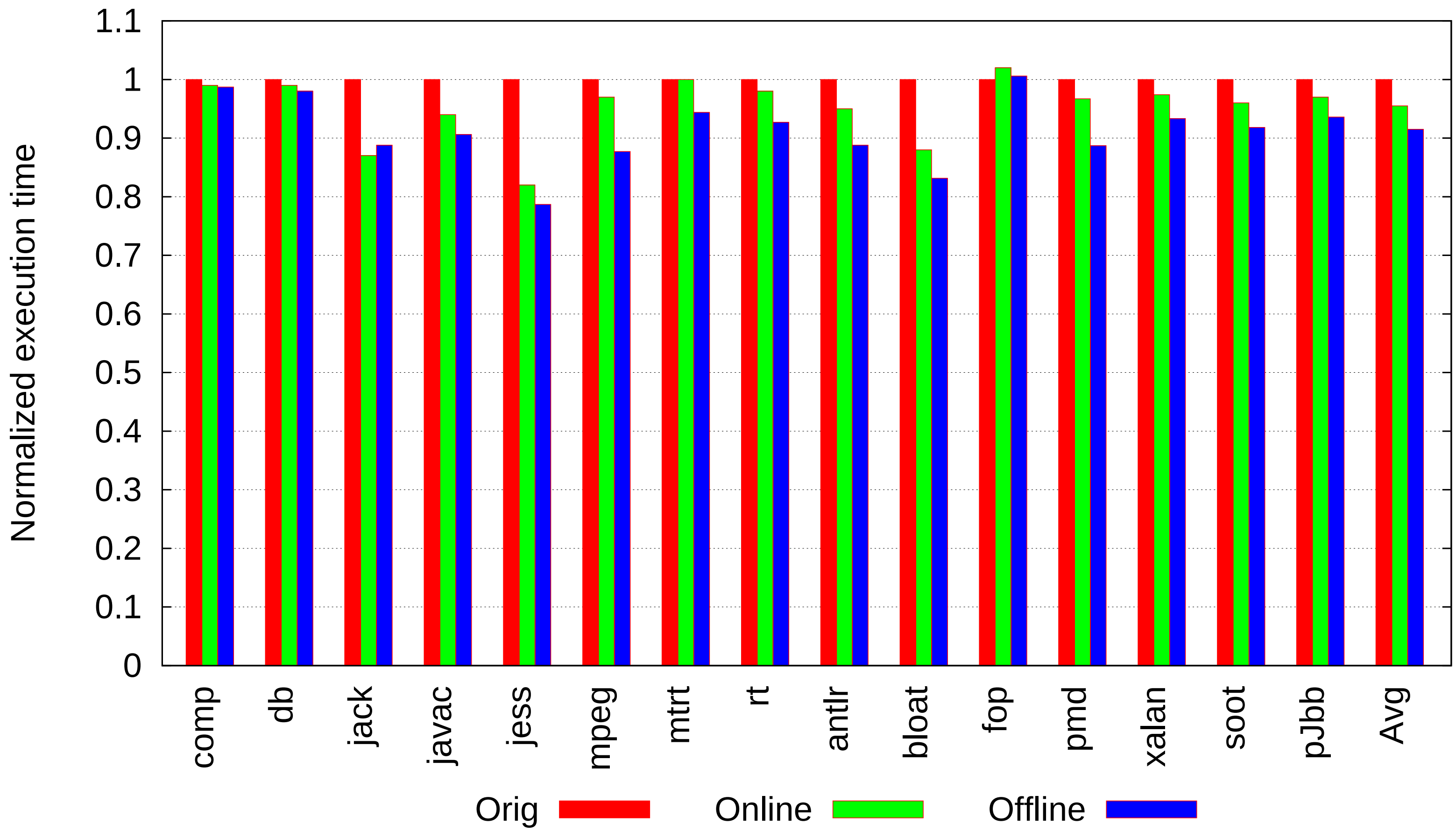
Offline Head Space Study



Experimental Set-up

- Benchmarks:
  - SPECjvm98 suite
  - Dacapo benchmarks: antlr, bloat, fop, pmd, xalan
  - Soot and PseudoJbb
- Test on Athlon 1.4G, 1GB memory, Debian Linux kernel 2.6.9
- Average of the middle 11 in 15 runs

Recompilation Results



Whole Execution Time Reduction (incl. all overhead)		Average	Up to
Offline	Use training runs	**8.7%**	**21%**
Online	Use HW pattern info.	**4.4%**	**18%**

Whole Overhead

Benchmark	Overhead (%)	Benchmark	Overhead (%)
compress	2.02	antlr	2.12
db	1.39	bloat	1.65
jack	1.71	fop	1.69
javac	1.13	pmd	1.70
jess	0.49	xalan	1.07
mpegaudio	1.76	soot	1.85
mtrt	0.82	PseudoJbb	0.77
raytrace	1.30	**Average**	**1.43**

- The "overhead" includes all sources from hardware monitoring, pattern construction, information analysis, and building control events to adaptive engine

Conclusions

- Understanding repetitive program behaviour and exploring phases in program execution is important
- We implemented a technique for determining program phases from hardware data
- We applied the phase information in adaptive recomplication
- Hardware information can be used in a wide range of areas
  - Runtime profiling, selecting GC points
  - Program understanding, system reconfiguration, instruction/data relocation and prefetch ...

Future Work

- Test other hardware events/combinations
- Use offline analysis results for repeatable executions
- Attach hardware variation with software structures
- Advanced static analysis can be helpful
- Develop other adaptive applications

Thank you!

Questions?