Tracking Performance of the Graal Compiler on Public Benchmarks

Lubomír Bulej [1] François Farquet [2] Vojtěch Horký [1]
Michele Tucci [1] Petr Tůma [1]

[1]Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University

[2]Oracle Labs Zürich

2018 – 2021

Department of Distributed and Dependable Systems

Disclaimer

Development Versions

Performance and other measurements used in this presentation are collected using **development** versions of the software involved.
As such, they do **not** represent product performance.

Modified Benchmarks

Benchmarks used to collect the measurements **were often modified** to facilitate integration into the measurement infrastructure.
None of the benchmark results are standard benchmark scores.

Platform Specific

Measurements are **platform specific**. Platform information was omitted for brevity, contact us if you need more details.

... and we are only human

The data may be influenced by mistakes we are not aware of.

Outline

About Graal Compiler

A just-in-time compiler for Java written in Java

- Functions as the last tier compiler
- Partial escape analysis and speculative optimizations

Part of a larger ecosystem surrounding the JVM

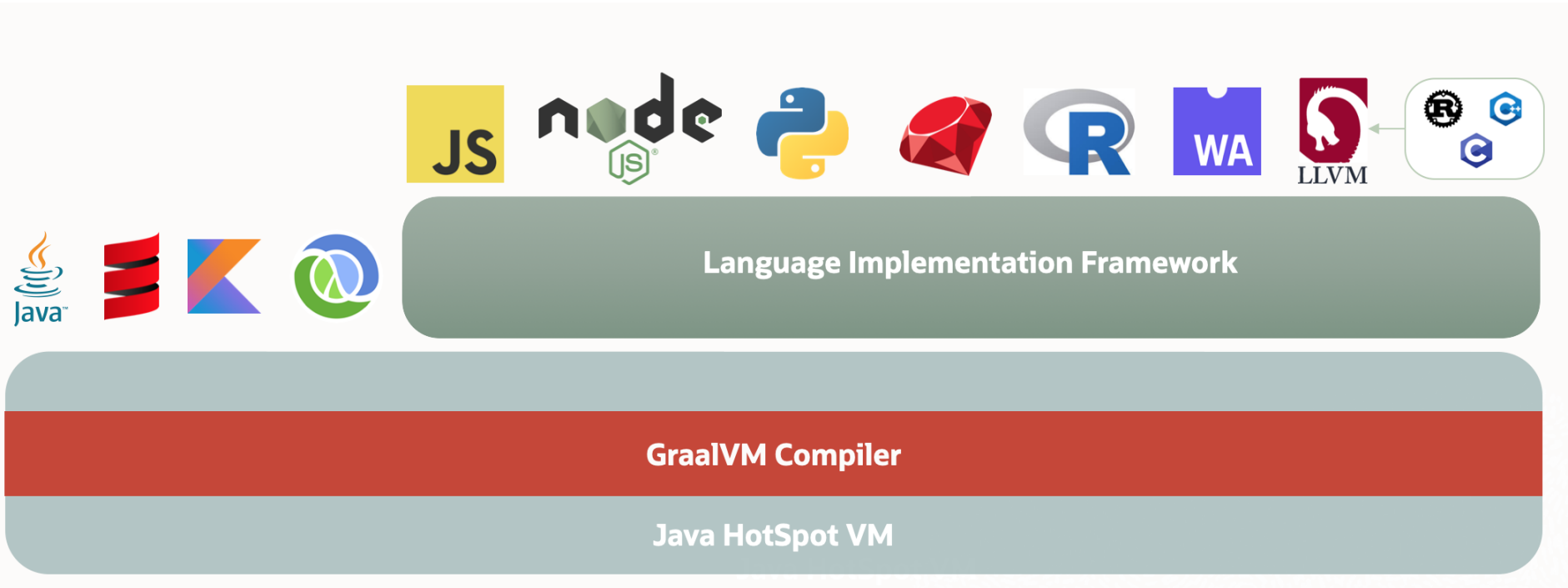
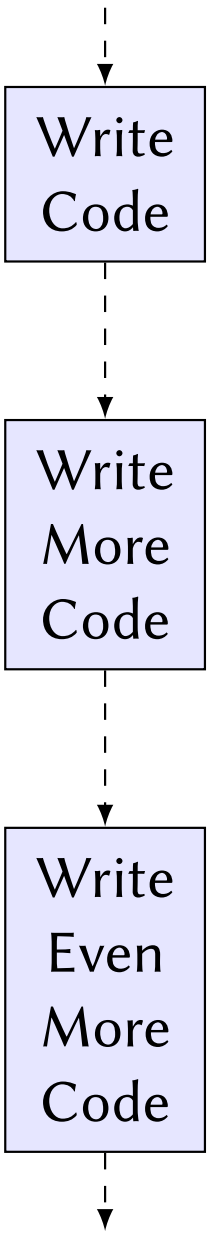


Image from `https://www.graalvm.org`

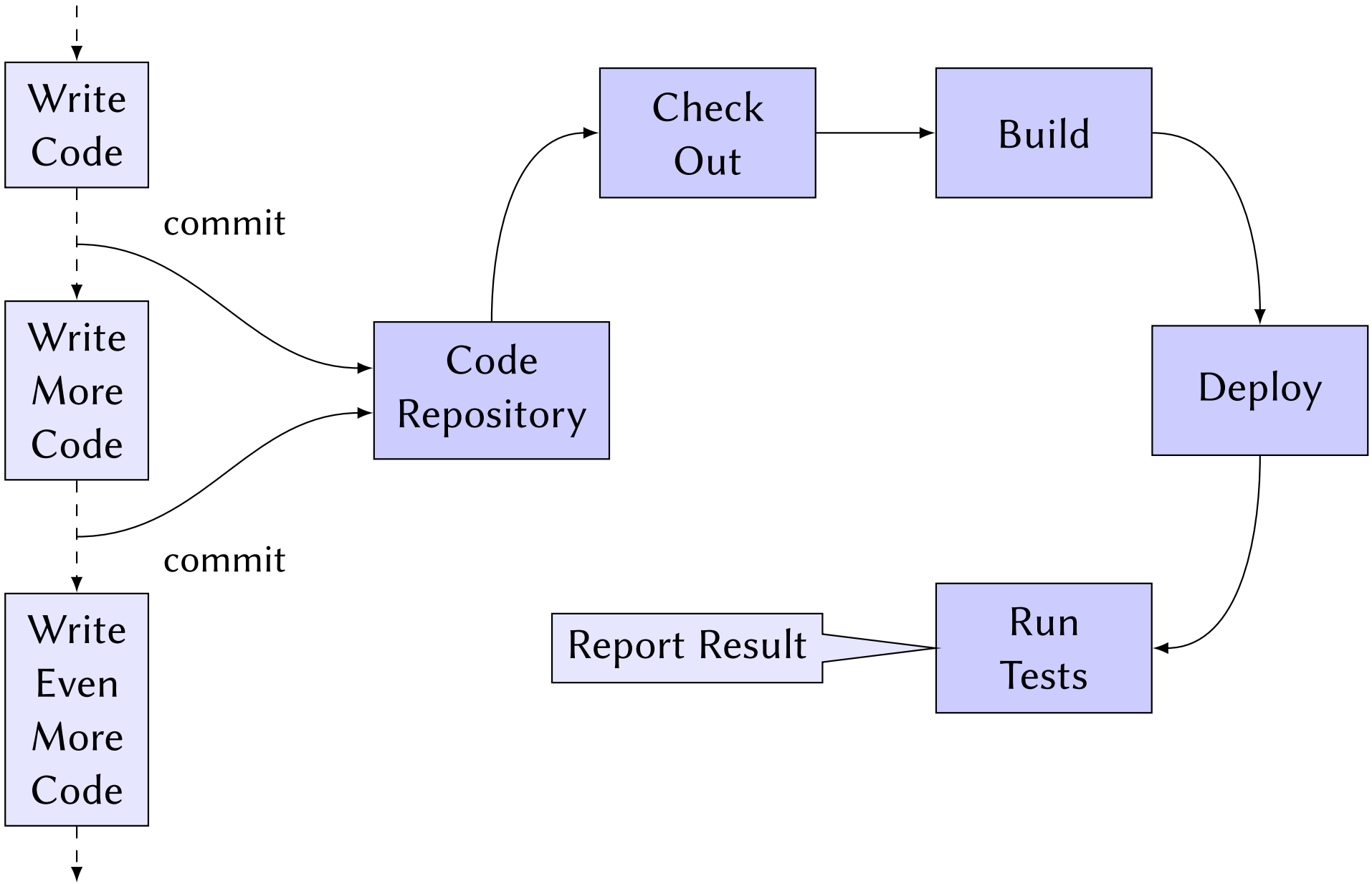
Performance Testing Goal ?

Make performance testing roughly the same as standard (functional) regression testing.

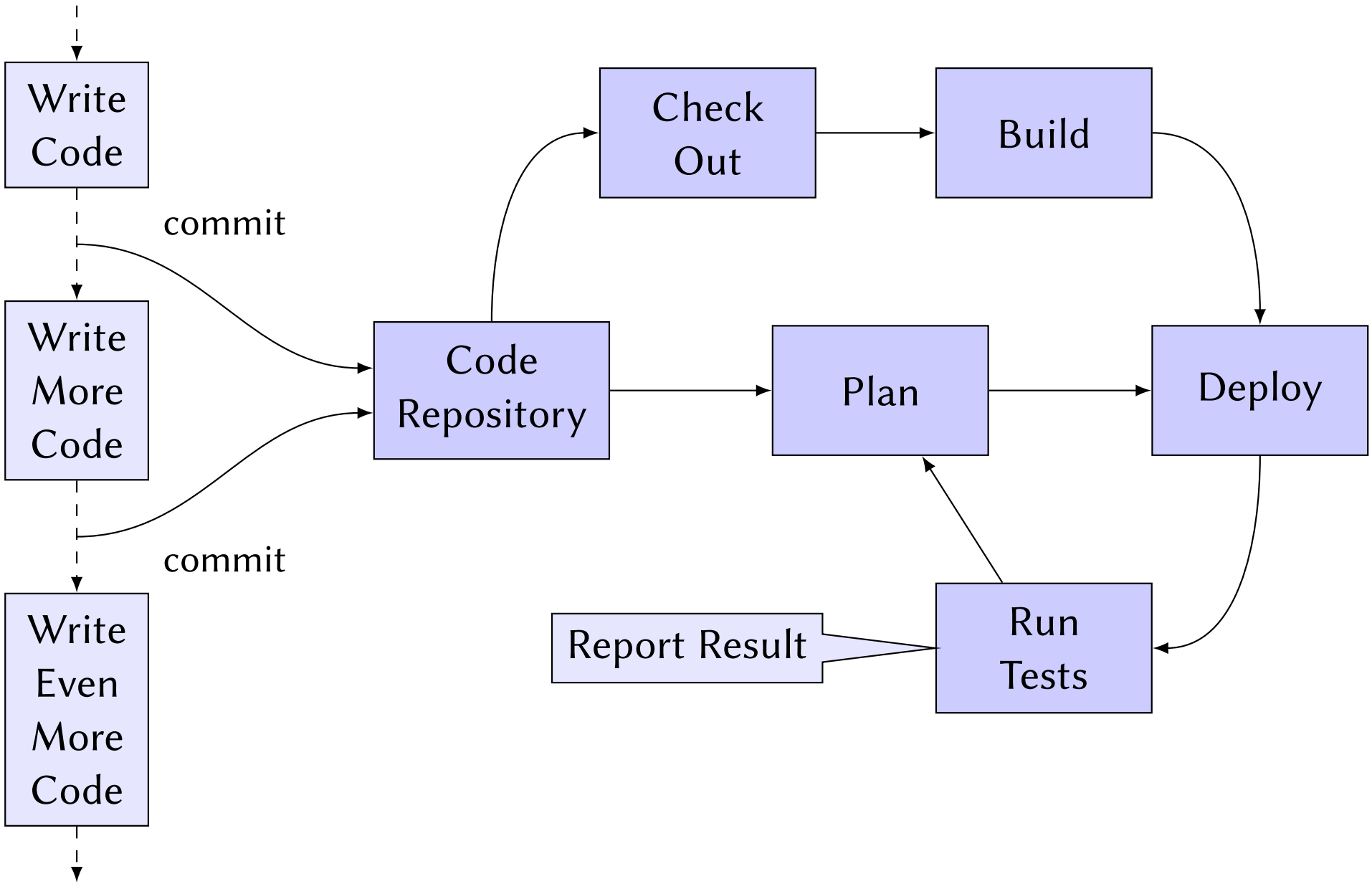
Performance Testing Workflow



Performance Testing Workflow

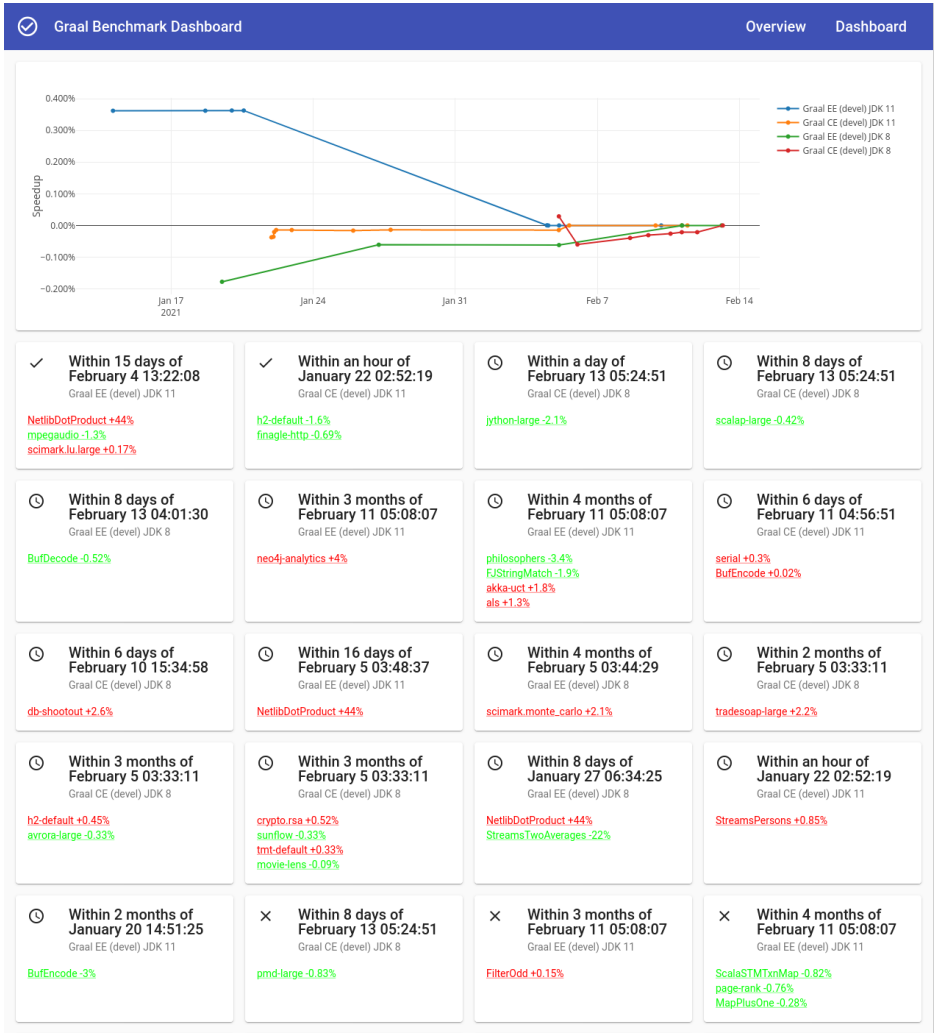


Performance Testing Workflow



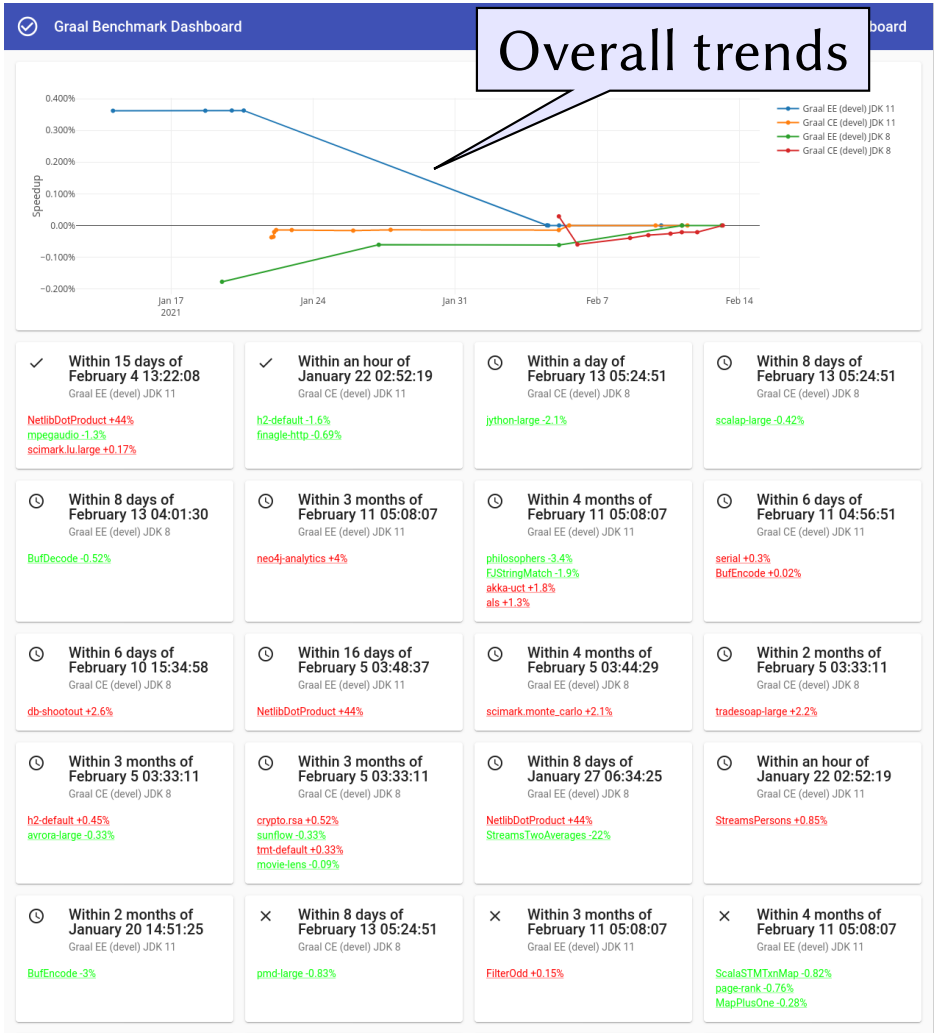
Performance Dashboard

https://graal.d3s.mff.cuni.cz



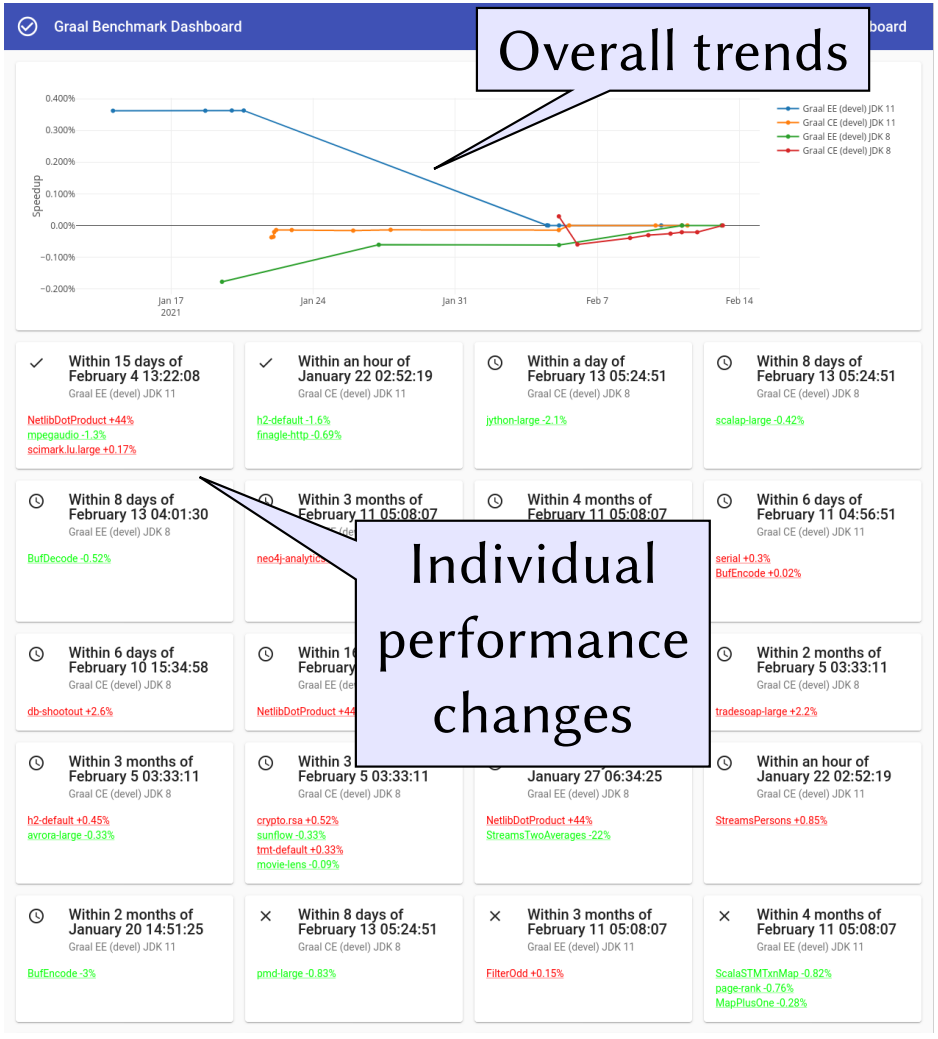
Performance Dashboard

https://graal.d3s.mff.cuni.cz



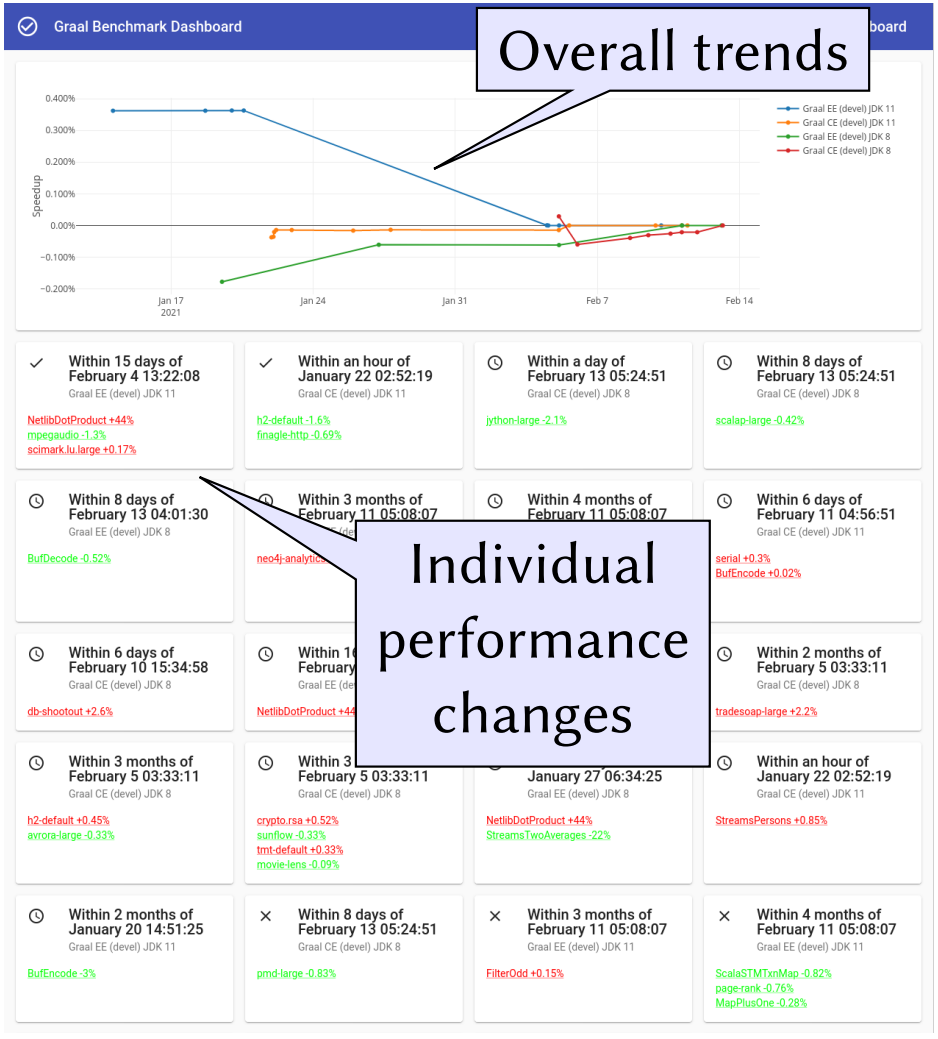
Performance Dashboard

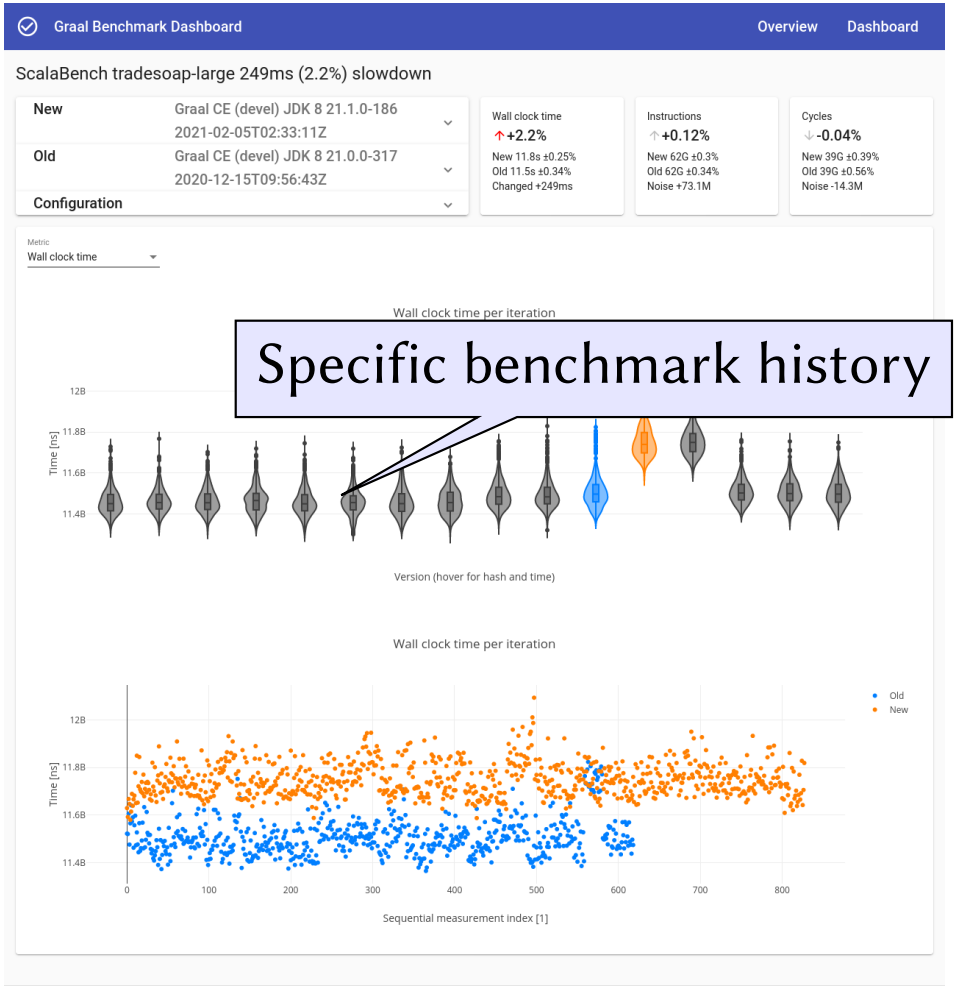
https://graal.d3s.mff.cuni.cz



Performance Dashboard

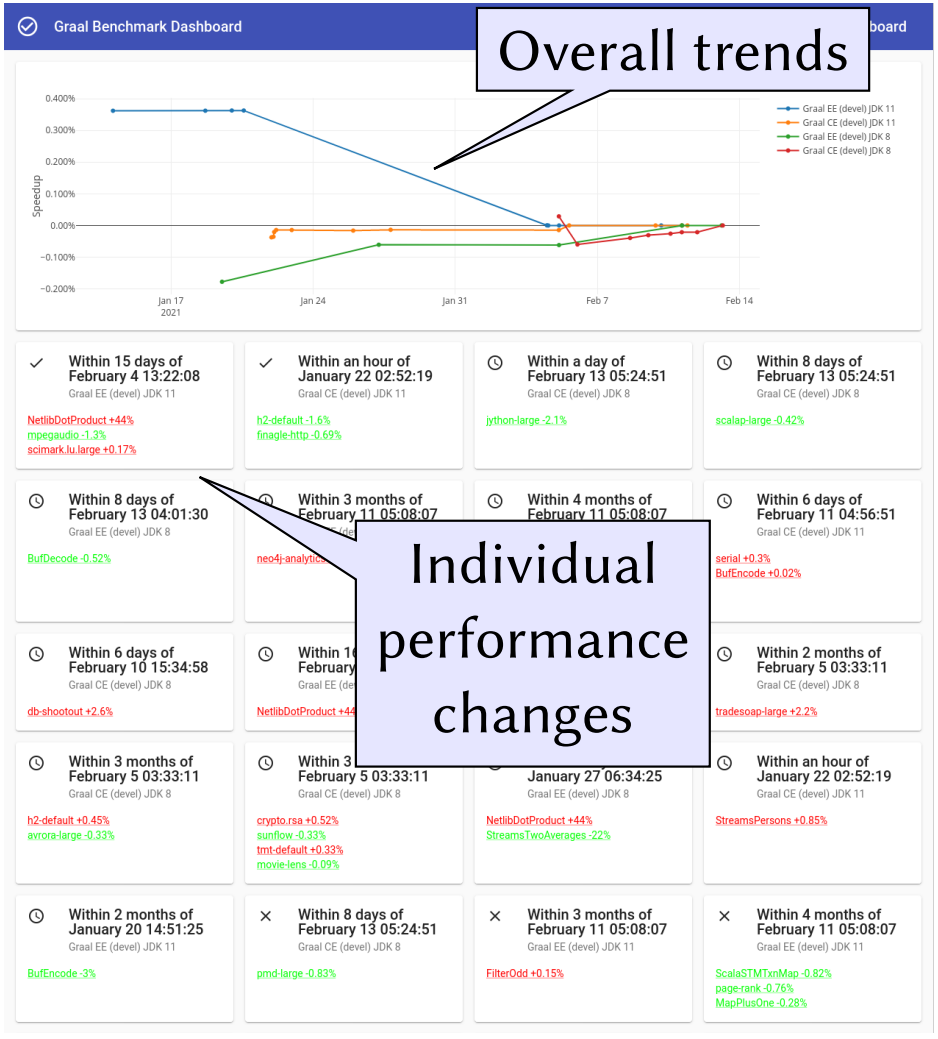
https://graal.d3s.mff.cuni.cz

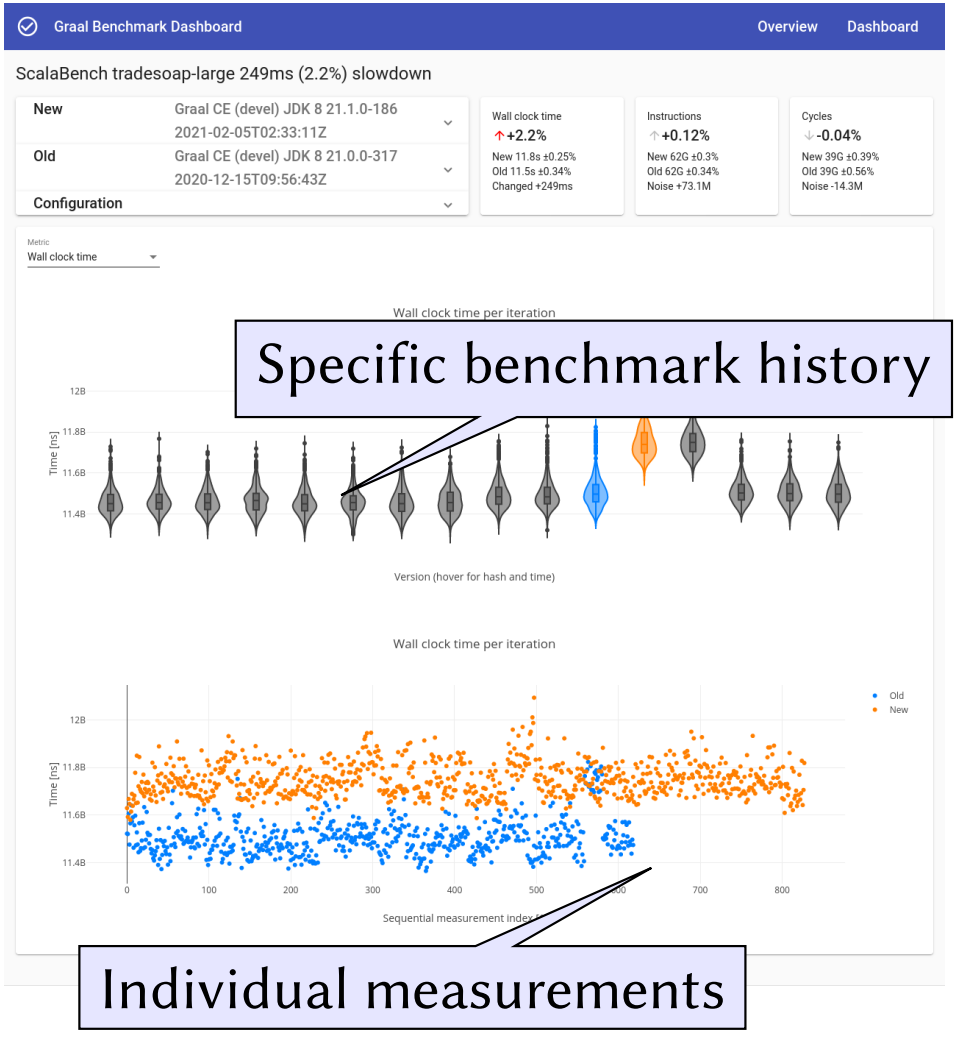




Performance Dashboard

https://graal.d3s.mff.cuni.cz





Dashboard Internals I

How to execute the measurements ?

- Resource sharing and background load matter
- Repetition count is determined on the fly
- Need more than latest software version
- Faulty setup may remain invisible

What we do

- Use dedicated hardware infrastructure
  - Multiple servers with equivalent parameters
  - No other load than the benchmarks
- Proprietary software to coordinate measurements
- Iterative selection of versions to measure

Dashboard Internals II

When to fail the test ?

- Noisy measurements
- Change can be legitimate
- Absolute performance requirements not given

What we do

- Compare performance of neighboring versions
- Focus on low false positive rate
  - Iterative measurement planning
  - Observing multiple metrics together
- Alongside commit pipeline but not blocking

Dashboard Internals III

Platforms

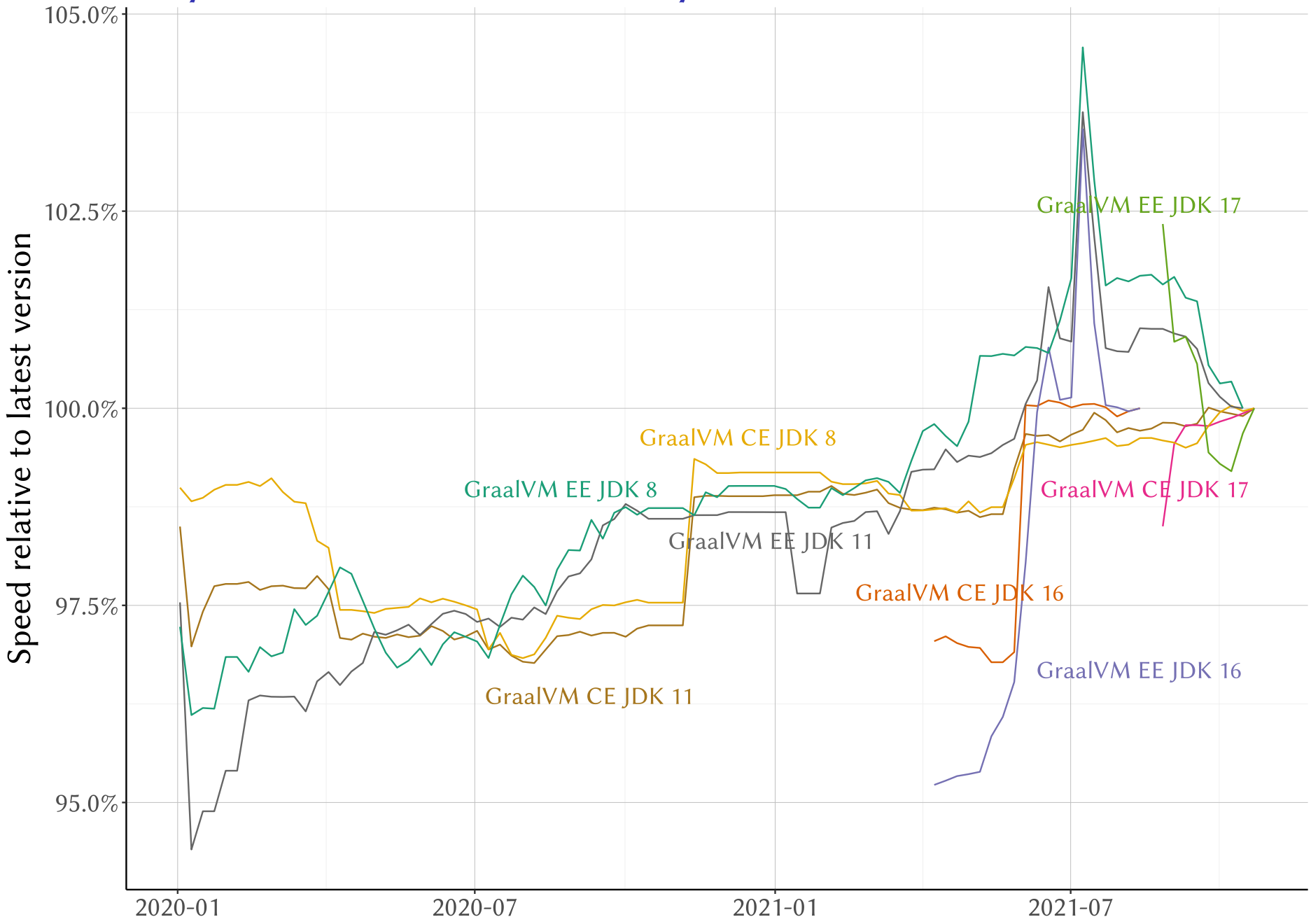
- GraalVM CE/EE with OpenJDK/HotSpot JDK 8/11/17 using JIT/AOT
- Only top level merge commits into master
- ... around 7800 last year

Benchmarks

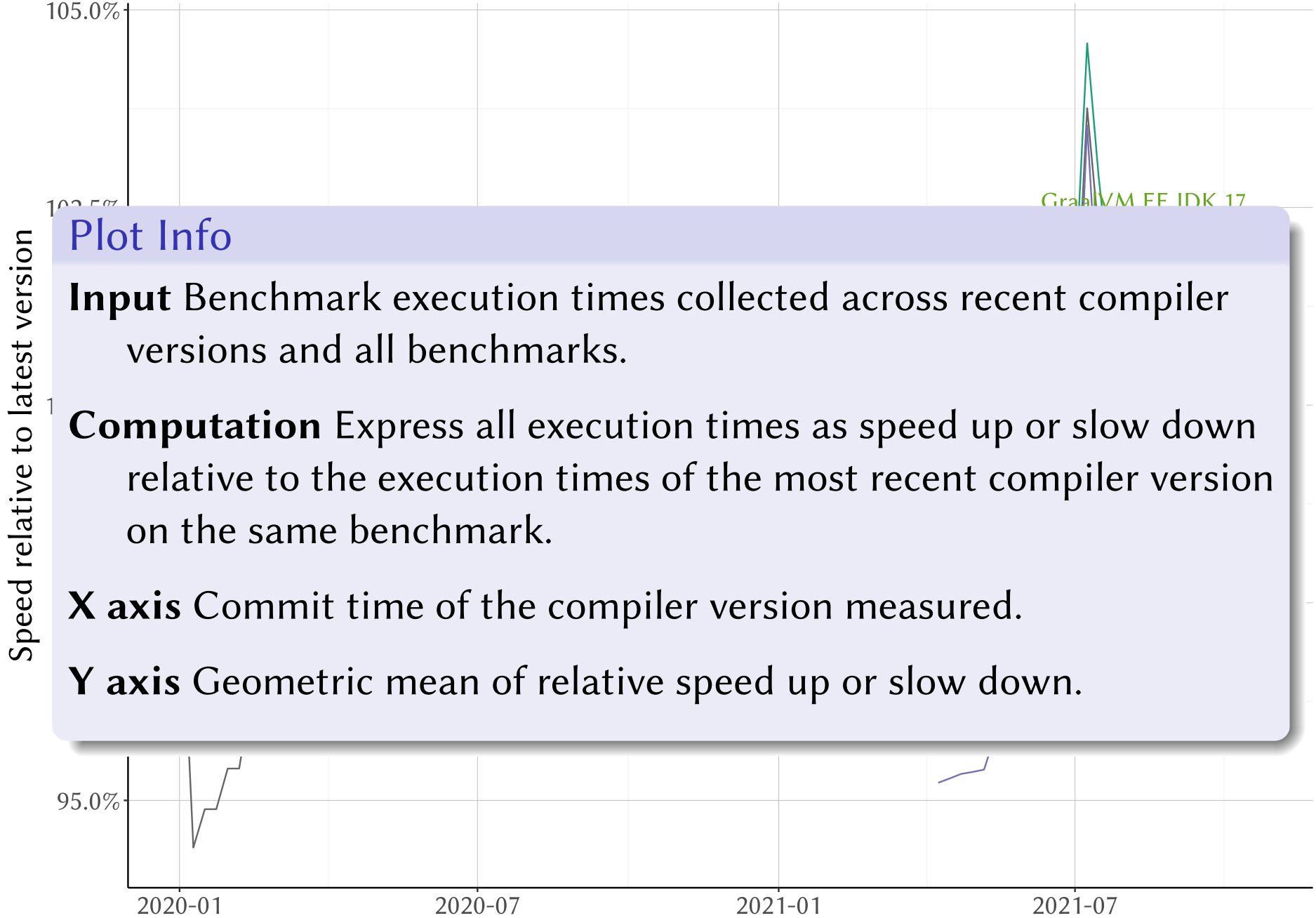
- ScalaBench (includes DaCapo) `https://scalabench.org`
- SPECjvm2008 (non-compliant) `https://spec.org/jvm2008`
- Renaissance 0.10 to 0.13 `https://renaissance.dev`
- Plus internal microbenchmarks
- ... around 150 workloads

Outline

Summary Performance History



Summary Performance History



Plot Info

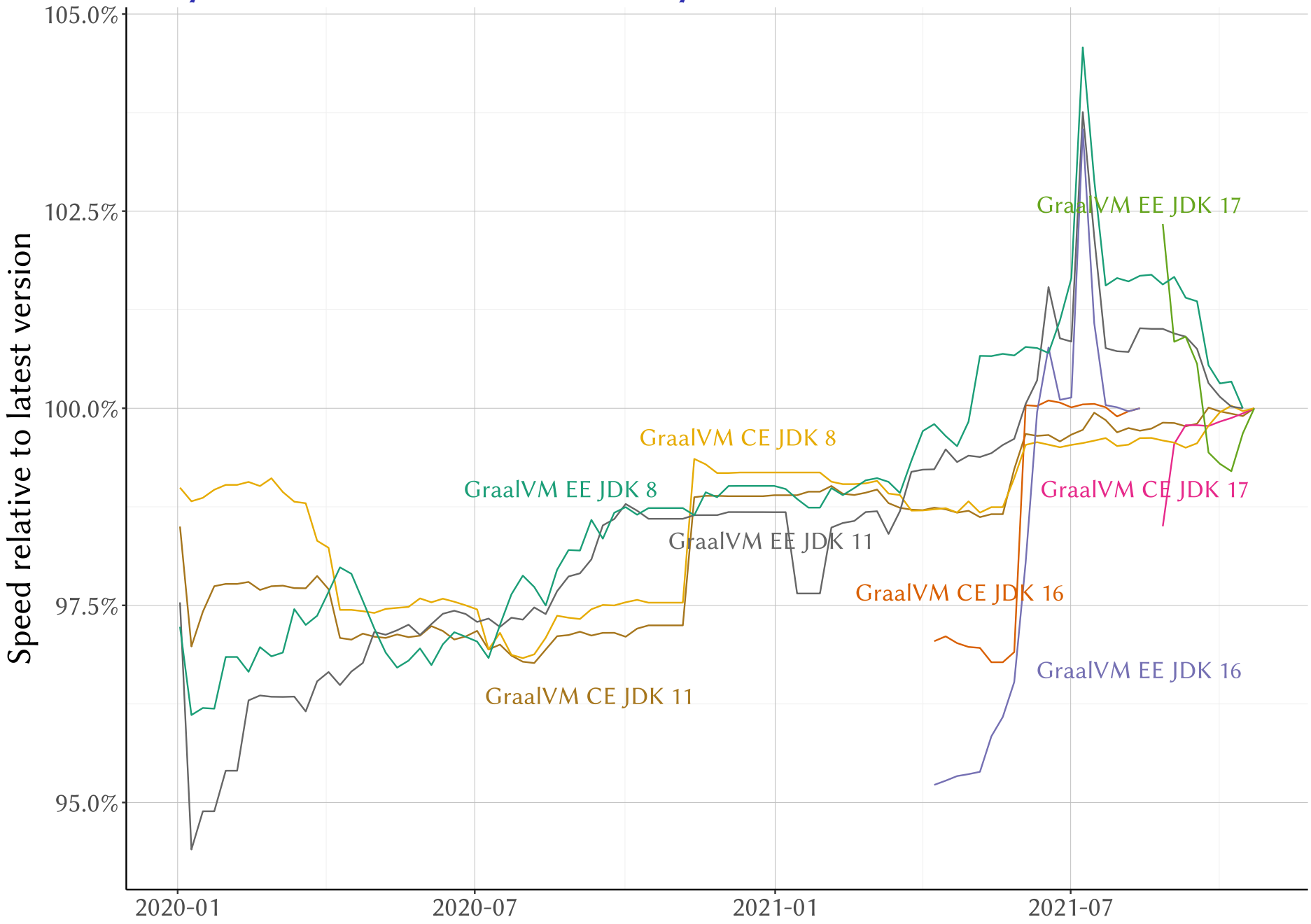
Input Benchmark execution times collected across recent compiler versions and all benchmarks.

Computation Express all execution times as speed up or slow down relative to the execution times of the most recent compiler version on the same benchmark.

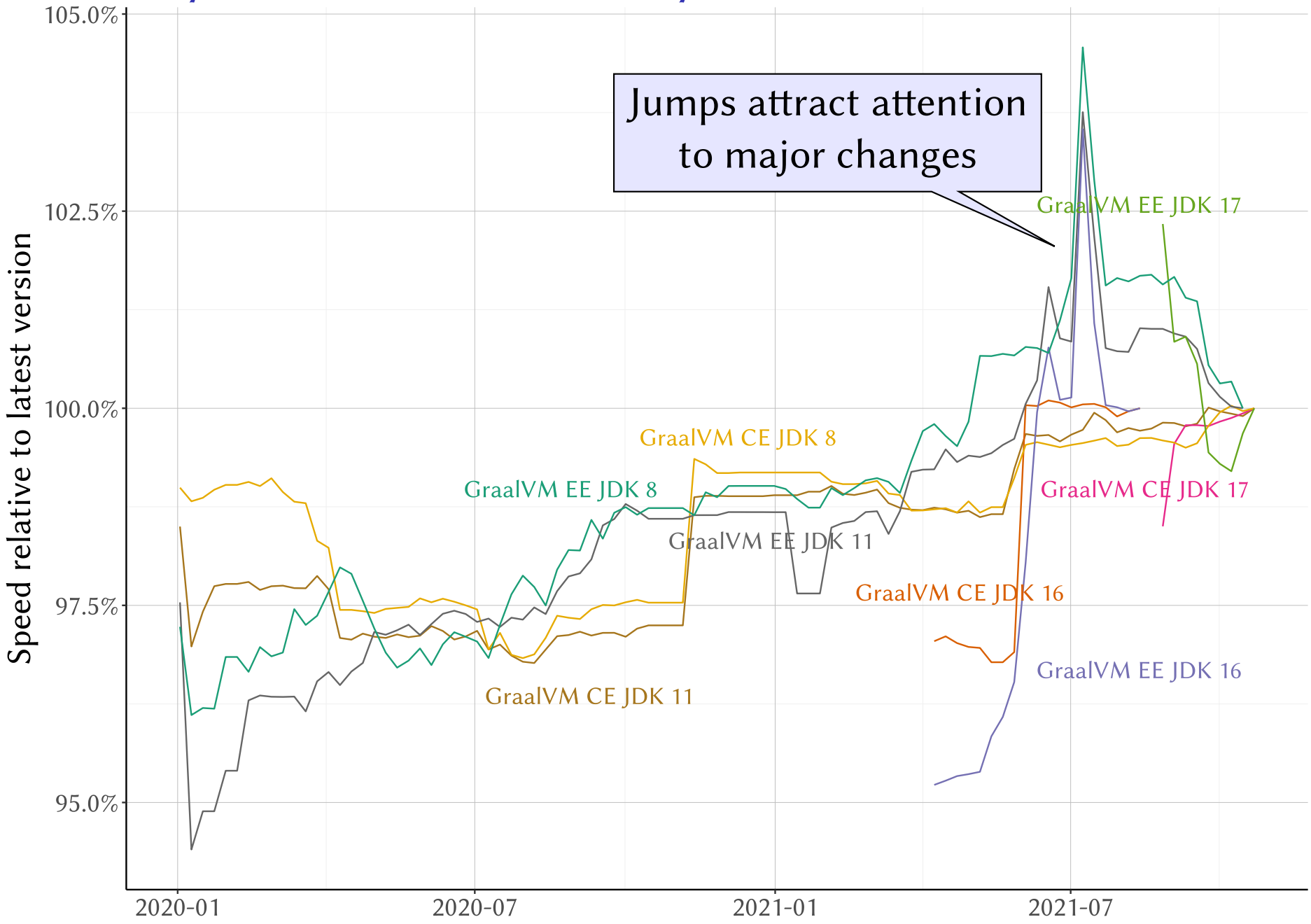
X axis Commit time of the compiler version measured.

Y axis Geometric mean of relative speed up or slow down.

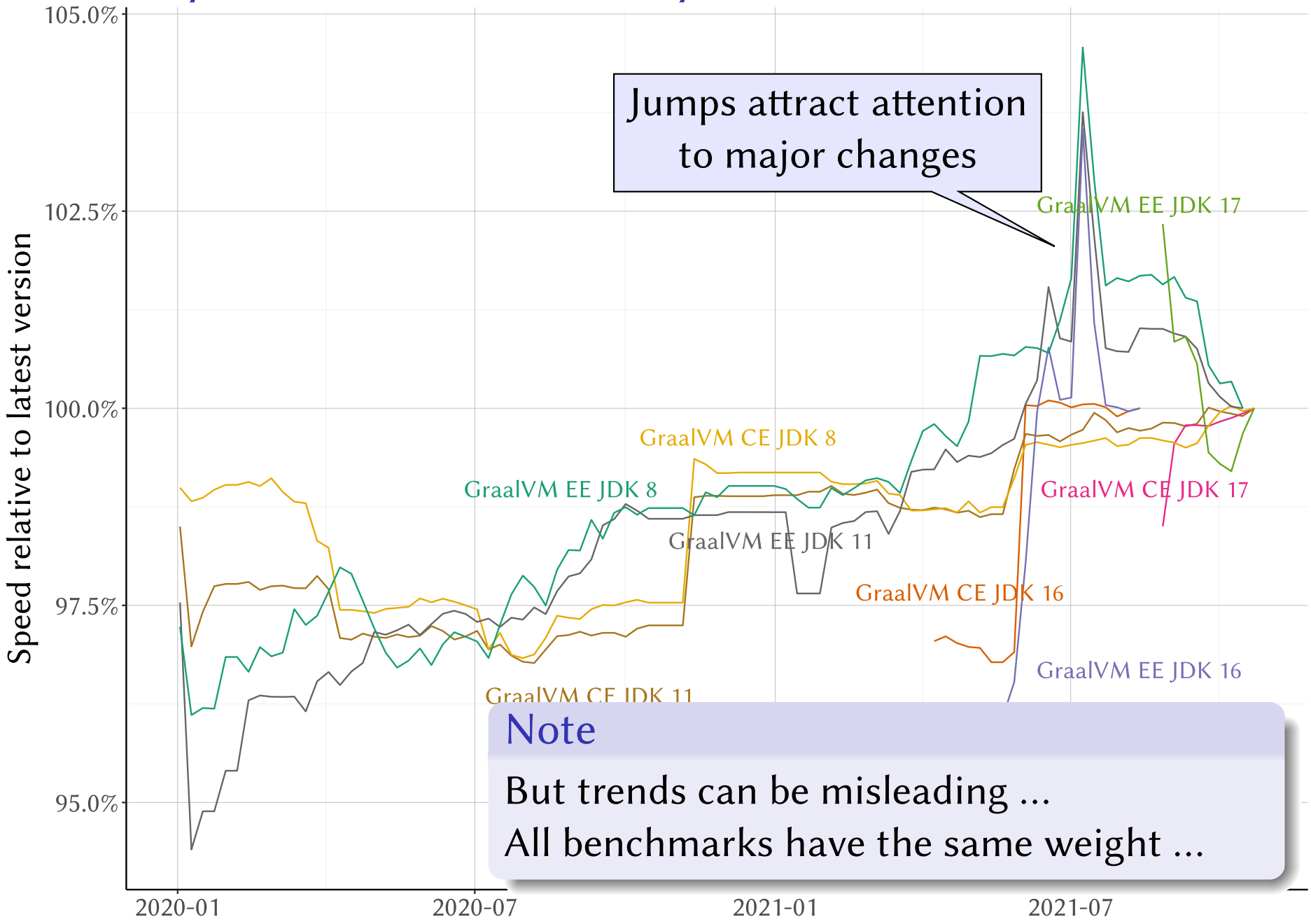
Summary Performance History



Summary Performance History



Summary Performance History

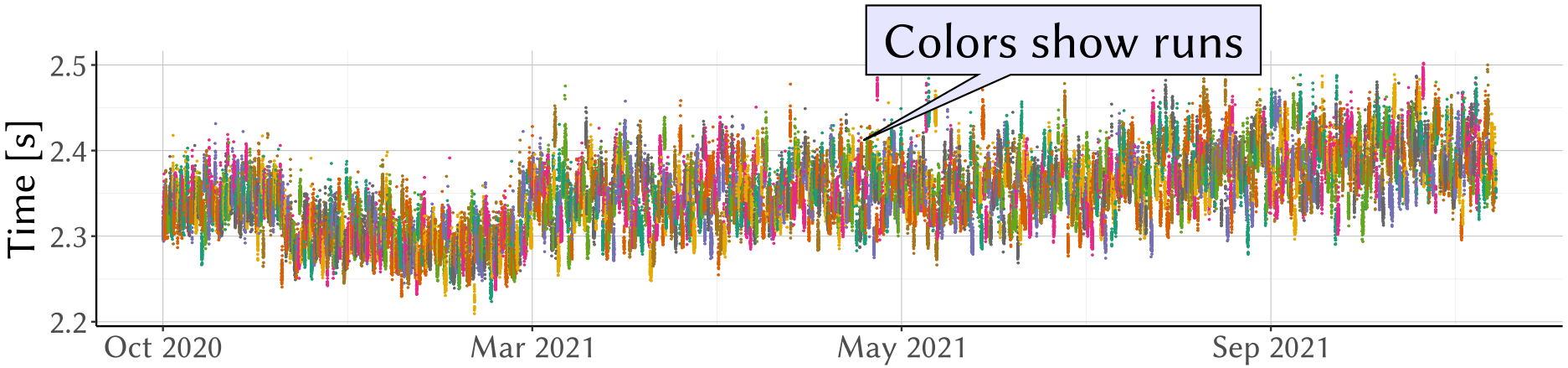


Note

But trends can be misleading ...

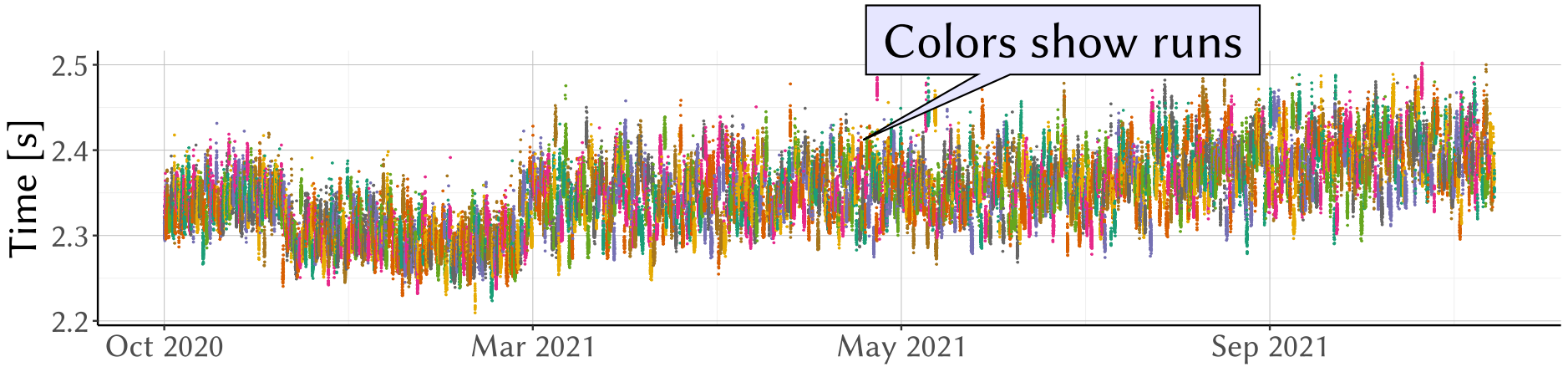
All benchmarks have the same weight ...

Detecting Individual Changes



https://doi.org/10.1007/s10515-015-0188-0

Detecting Individual Changes



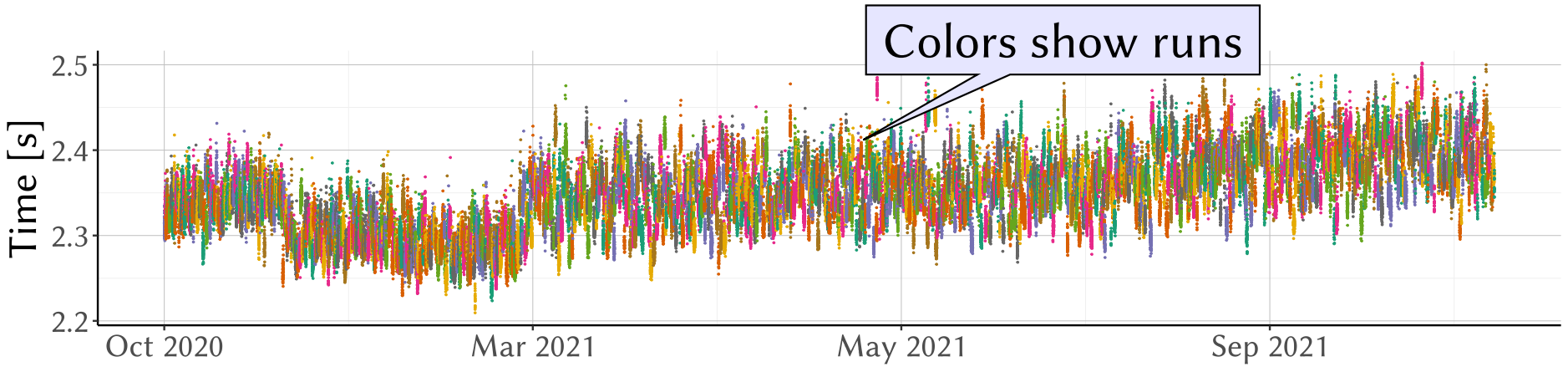
A time series change point detection problem with a few twists

- We have more correlated time series rather than just one
- We can add more data points to any version if required
- Data points are in fact hierarchical sets from runs
- We are more interested in changes near series end
- Almost no assumptions about data distribution

---

https://doi.org/10.1007/s10515-015-0188-0

Detecting Individual Changes



A time series change point detection problem with a few twists

- We have more correlated time series rather than just one
- We can add more data points to any version if required
- Data points are in fact hierarchical sets from runs
- We are more interested in changes near series end
- Almost no assumptions about data distribution

We use bootstrap confidence intervals of mean differences

---

https://doi.org/10.1007/s10515-015-0188-0

Detected Changes In Numbers

What share of versions have changes and how reliably are they detected ?

Renaissance 0.10			
bench	R	D	I
aka-uct	1%	100%	0%
als	5%	100%	0%
chi-sqr	2%	100%	0%
db-shot	2%		
dec-tre	2%	100%	0%
dotty	5%		
fin-chi	1%	100%	0%
fin-htt	3%	100%	0%
fj-kmns	5%	100%	0%
fut-gen	0%		
gauss	1%		
log-reg	6%	100%	0%
mne	5%	100%	0%
mov-len	6%		
nai-bay	2%		
neo-ana	4%	100%	0%
pg-rank	1%	100%	0%
par-mne	4%	100%	0%
philos	2%		
reactr	2%	100%	0%

rx-scrb	4%	100%	0%
sc-doku	1%	50%	50%
sc-kmns	6%		
sc-stmb	1%		
scrb	5%	100%	0%
ScalaBench (with DaCapo)			
bench	R	D	I
appar-d	3%	100%	0%
avror-l	1%		
batik-s	3%	67%	33%
eclps-s	1%		
factr-d	1%	100%	0%
fop-d	2%	100%	0%
h2-d	2%	100%	0%
jythn-l	1%	100%	0%
kiama-d	2%	89%	11%
luidx-d	1%	100%	0%
lusrc-l	2%	50%	44%
pmd-l	3%	67%	33%
scc-l	1%	100%	0%
scdoc-l	1%	100%	0%
scp-l	2%	17%	83%

scrfm-h	2%	50%	50%
scxb-h	2%	92%	8%
specs-l	1%	100%	0%
sunfl-l	2%	100%	0%
tmt-d	3%	25%	75%
trdb-d	1%	100%	0%
trds-l	2%	89%	11%
xalan-l	2%	90%	10%
SPECjvm2008 (modified)			
bench	R	D	I
cmp.cmp	2%		
cmp.sun	2%		
compr	4%	75%	25%
cry.aes	4%	100%	0%
cry.rsa	2%	100%	0%
cry.sgn	4%	75%	25%
derby	1%	60%	40%
mpega	4%	100%	0%
sci.ffl	1%	67%	33%
sci.lul	1%	50%	0%
sci.mtc	3%	88%	12%
sci.sol	3%	100%	0%

sci.spl	4%	100%	0%
serial	2%	75%	25%
sunflow	3%	100%	0%
xml.trn	3%	50%	50%
xml.val	2%	75%	25%
Internal Micros			
bench	R	D	I
StrDev	4%	33%	67%
SFndNeg	3%	36%	50%
SFldSum	3%	25%	50%
SForSum	3%	42%	11%
SMapRed	3%	43%	21%
STwoAvg	4%	60%	30%
TSP	4%	100%	0%
TxtSDF	2%	80%	10%
TxtRDD	2%	100%	0%
WrdCnt	1%	100%	0%
BufDec	6%	78%	15%
BufEnc	6%	88%	12%
ChrCnt	2%	100%	0%
ChrHis	3%	73%	20%
FJHis	7%	100%	0%

FJStr	7%	100%	0%
FltOdd	12%	50%	50%
FndNgt	3%	62%	8%
FntNgtR	2%	50%	0%
FldSum	3%	100%	0%
FldSumR	0%	0%	33%
ForSum	1%	50%	0%
ForSumR	2%	12%	75%
GrpRem	5%	85%	0%
MapOne	7%	76%	14%
NetDot	3%	57%	0%
NetEig	2%	62%	25%
Reduce	1%	50%	50%
STMLst	2%	50%	0%
STMMap	3%	100%	0%
Scan	1%	43%	57%
SrtRDD	2%	70%	30%
StdDev	3%	25%	44%
StrCnt	2%	50%	50%
StrDem	2%	50%	0%
StrPer	4%	93%	0%

R - versions with changes

D - manually confirmed

I - invalid situations

Detected Changes In Numbers

What share of versions have changes and how reliably are they detected ?

Renaissance 0.10

bench	R	D	I
aka-uct	1%	100%	0%
als	5%	100%	0%
chi-sqr	2%	100%	0%
db-shot	2%		
dec-tre	2%	100%	0%
dotty	5%		
fin-chi	1%	100%	0%
fin-htt	3%	100%	0%
fj-kmns	5%	100%	0%
fut-gen	0%		
gauss	1%		
log-reg	6%	100%	0%
mne	[illegible]	100%	0%
[illegible]	[illegible]		
[illegible]	[illegible]		
[illegible]	[illegible]		
pg rank	1%	100%	0%
par-mne	4%	100%	0%
philos	2%		
reactr	2%	100%	0%

rx-scrb	4%	100%	0%
sc-doku	1%	50%	50%
sc-kmns	6%		
sc-stmb	1%		
scrb	5%	100%	0%

ScalaBench (with DaCapo)

bench	R	D	I
appar-d	3%	100%	0%
avror-l	1%		
batik-s	3%	67%	33%
eclps-s	1%		
factr-d	1%	100%	0%
fop-d	2%	100%	0%
h2-d	2%	100%	0%
jythn-l	1%	100%	0%
[illegible]d	2%	89%	11%
[illegible]d	1%	100%	0%
[illegible]l	2%	50%	44%
pmd-l	3%	67%	33%
scc-l	1%	100%	0%
scdoc-l	1%	100%	0%
scp-l	2%	17%	83%

scrfm-h	2%	50%	50%
scxb-h	2%	92%	8%
specs-l	1%	100%	0%
sunfl-l	2%	100%	0%
tmt-d	3%	25%	75%
trdb-d	1%	100%	0%
trds-l	2%	89%	11%
xalan-l	2%	90%	10%

SPECjvm2008 (modified)

bench	R	D	I
cmp.cmp	2%		
cmp.sun	2%		
compr	4%	75%	25%
cry.aes	4%	100%	0%
cry.rsa	2%	100%	0%
cry.sgn	4%	75%	25%
derby	1%	60%	40%
mpega	4%	100%	0%
sci.ffl	1%	67%	33%
sci.lul	1%	50%	0%
sci.mtc	3%	88%	12%
sci.sol	3%	100%	0%

sci.spl	4%	100%	0%
serial	2%	75%	25%
sunflow	3%	100%	0%
xml.trn	3%	50%	50%
xml.val	2%	75%	25%

Internal Micros

bench	R	D	I
StrDev	4%	33%	67%
SFndNeg	3%	36%	50%
SFldSum	3%	25%	50%
SForSum	3%	42%	11%
SMapRed	3%	43%	21%
STwoAvg	4%	60%	30%
TSP	4%	100%	0%
TxtSDF	2%	80%	10%
TxtRDD	2%	100%	0%
WrdCnt	1%	100%	0%
BufDec	6%	78%	15%
BufEnc	6%	88%	12%
ChrCnt	2%	100%	0%
ChrHis	3%	73%	20%
FJHis	7%	100%	0%

FJStr	7%	100%	0%
FltOdd	12%	50%	50%
FndNgt	3%	62%	8%
FntNgtR	2%	50%	0%
FldSum	3%	100%	0%
FldSumR	0%	0%	33%
ForSum	1%	50%	0%
ForSumR	2%	12%	75%
GrpRem	5%	85%	0%
MapOne	7%	76%	14%
NetDot	3%	57%	0%
NetEig	2%	62%	25%
Reduce	1%	50%	50%
STMLst	2%	50%	0%
STMMap	3%	100%	0%
Scan	1%	43%	57%
SrtRDD	2%	70%	30%
StdDev	3%	25%	44%
StrCnt	2%	50%	50%
StrDem	2%	50%	0%
StrPer	4%	93%	0%

Most benchmarks exhibit changes

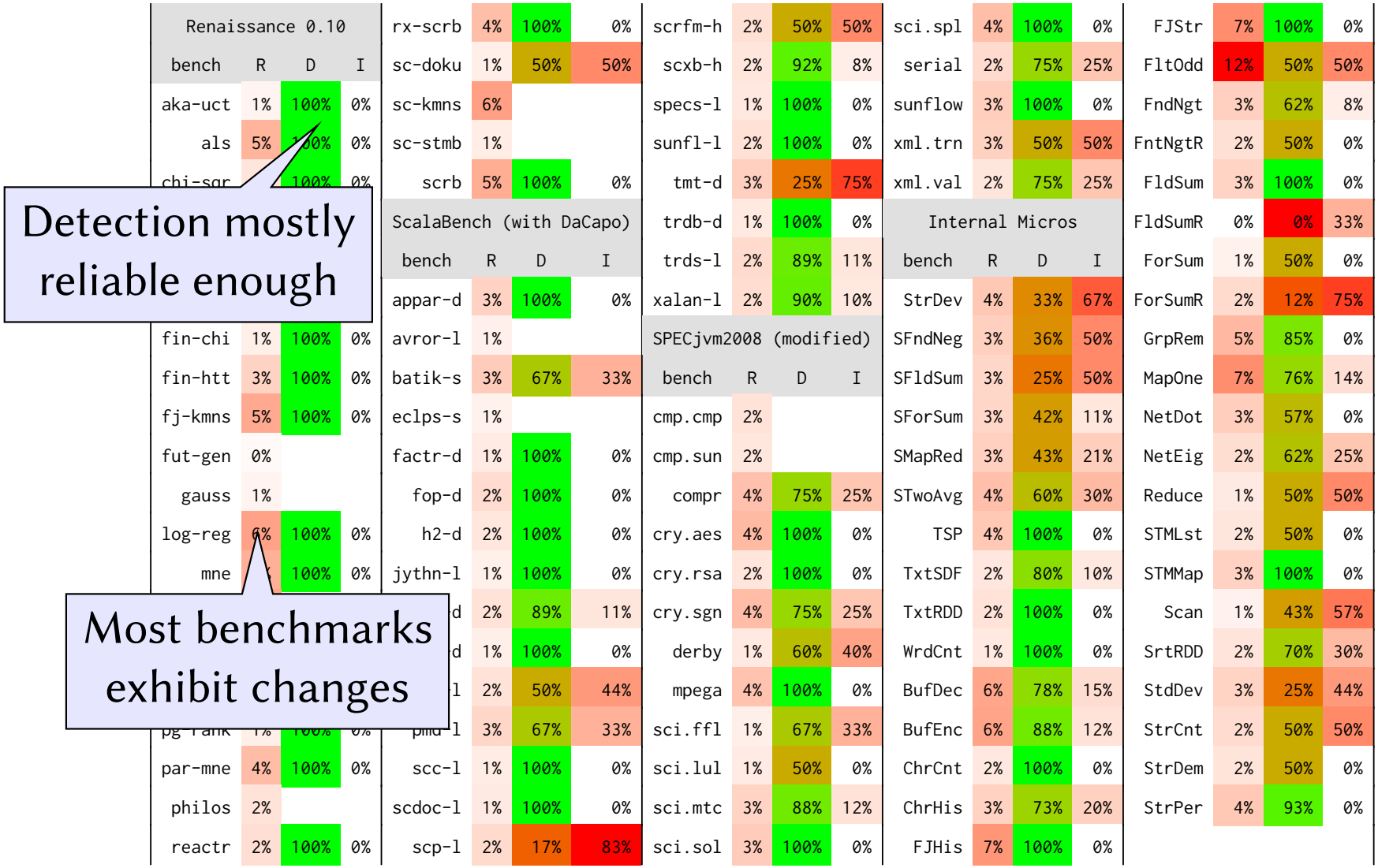
R - versions with changes

D - manually confirmed

I - invalid situations

Detected Changes In Numbers

What share of versions have changes and how reliably are they detected ?



Renaissance 0.10			
bench	R	D	I
aka-uct	1%	100%	0%
als	5%	100%	0%
chi-sqr		100%	0%
fin-chi	1%	100%	0%
fin-htt	3%	100%	0%
fj-kmns	5%	100%	0%
fut-gen	0%		
gauss	1%		
log-reg	6%	100%	0%
mne		100%	0%
pg rank	1%	100%	0%
par-mne	4%	100%	0%
philos	2%		
reactr	2%	100%	0%

rx-scrb	4%	100%	0%
sc-doku	1%	50%	50%
sc-kmns	6%		
sc-stmb	1%		
scrb	5%	100%	0%

ScalaBench (with DaCapo)			
bench	R	D	I
appar-d	3%	100%	0%
avror-l	1%		
batik-s	3%	67%	33%
eclps-s	1%		
factr-d	1%	100%	0%
fop-d	2%	100%	0%
h2-d	2%	100%	0%
jythn-l	1%	100%	0%
d	2%	89%	11%
d	1%	100%	0%
l	2%	50%	44%
pmd-l	3%	67%	33%
scc-l	1%	100%	0%
scdoc-l	1%	100%	0%
scp-l	2%	17%	83%

scrfm-h	2%	50%	50%
scxb-h	2%	92%	8%
specs-l	1%	100%	0%
sunfl-l	2%	100%	0%
tmt-d	3%	25%	75%
trdb-d	1%	100%	0%
trds-l	2%	89%	11%
xalan-l	2%	90%	10%

SPECjvm2008 (modified)			
bench	R	D	I
cmp.cmp	2%		
cmp.sun	2%		
compr	4%	75%	25%
cry.aes	4%	100%	0%
cry.rsa	2%	100%	0%
cry.sgn	4%	75%	25%
derby	1%	60%	40%
mpega	4%	100%	0%
sci.ffl	1%	67%	33%
sci.lul	1%	50%	0%
sci.mtc	3%	88%	12%
sci.sol	3%	100%	0%

sci.spl	4%	100%	0%
serial	2%	75%	25%
sunflow	3%	100%	0%
xml.trn	3%	50%	50%
xml.val	2%	75%	25%

Internal Micros			
bench	R	D	I
StrDev	4%	33%	67%
SFndNeg	3%	36%	50%
SFldSum	3%	25%	50%
SForSum	3%	42%	11%
SMapRed	3%	43%	21%
STwoAvg	4%	60%	30%
TSP	4%	100%	0%
TxtSDF	2%	80%	10%
TxtRDD	2%	100%	0%
WrdCnt	1%	100%	0%
BufDec	6%	78%	15%
BufEnc	6%	88%	12%
ChrCnt	2%	100%	0%
ChrHis	3%	73%	20%
FJHis	7%	100%	0%

FJStr	7%	100%	0%
FltOdd	12%	50%	50%
FndNgt	3%	62%	8%
FntNgtR	2%	50%	0%
FldSum	3%	100%	0%
FldSumR	0%	0%	33%
ForSum	1%	50%	0%
ForSumR	2%	12%	75%
GrpRem	5%	85%	0%
MapOne	7%	76%	14%
NetDot	3%	57%	0%
NetEig	2%	62%	25%
Reduce	1%	50%	50%
STMLst	2%	50%	0%
STMMap	3%	100%	0%
Scan	1%	43%	57%
SrtRDD	2%	70%	30%
StdDev	3%	25%	44%
StrCnt	2%	50%	50%
StrDem	2%	50%	0%
StrPer	4%	93%	0%

R - versions with changes

D - manually confirmed

I - invalid situations

Detected Changes In Numbers

What share of versions have changes and how reliably are they detected ?

Renaissance 0.10			
bench	R	D	I
aka-uct	1%	100%	0%
als	5%	100%	0%
chi-sqr		100%	0%
fin-chi	1%	100%	0%
fin-htt	3%	100%	0%
fj-kmns	5%	100%	0%
fut-gen	0%		
gauss	1%		
log-reg	6%	100%	0%
mne		100%	0%
par-mne	4%	100%	0%
philos	2%		
reactr	2%	100%	0%

rx-scrb	4%	100%	0%
sc-doku	1%	50%	50%
sc-kmns	6%		
sc-stmb	1%		
scrb	5%	100%	0%

ScalaBench (with DaCapo)			
bench	R	D	I
appar-d	3%	100%	0%
avror-l	1%		
batik-s	3%	67%	33%
eclps-s	1%		
factr-d	1%	100%	0%
fop-d	2%	100%	0%
h2-d	2%	100%	0%
jythn-l	1%	100%	0%
[illegible]d	2%	89%	11%
[illegible]d	1%	100%	0%
[illegible]l	2%	50%	44%
pmd-l	3%	67%	33%
scc-l	1%	100%	0%
scdoc-l	1%	100%	0%
scp-l	2%	17%	83%

scrfm-h	2%	50%	50%
scxb-h	2%	92%	8%
specs-l	1%	100%	0%
sunfl-l	2%	100%	0%
tmt-d	3%	25%	75%
trdb-d	1%	100%	0%
trds-l	2%	89%	11%
xalan-l	2%	90%	10%

SPECjvm2008 (modified)			
bench	R	D	I
cmp.cmp	2%		
cmp.sun	2%		
compr	4%	75%	25%
cry.aes	4%	100%	0%
cry.rsa	2%	100%	0%
cry.sgn	4%	75%	25%
derby	1%	60%	40%
mpega	4%	100%	0%
sci.ffl	1%	67%	33%
sci.lul	1%	50%	0%
sci.mtc	3%	88%	12%
sci.sol	3%	100%	0%

sci.spl	4%	100%	0%
serial	2%	75%	25%
sunflow	3%	100%	0%
xml.trn	3%	50%	50%
xml.val	2%	75%	25%

Internal Micros			
bench	R	D	I
StrDev	4%	33%	67%
SFndNeg	3%	36%	50%
SFldSum	3%	25%	[illegible]
SForSum	3%	42%	
SMapRed	3%	43%	
STwoAvg	4%	60%	
TSP	4%	100%	[illegible]
TxtSDF	2%	80%	10%
TxtRDD	2%	100%	0%
WrdCnt	1%	100%	0%
BufDec	6%	78%	15%
BufEnc	6%	88%	12%
ChrCnt	2%	100%	0%
ChrHis	3%	73%	20%
FJHis	7%	100%	0%

FJStr	7%	100%	0%
FltOdd	12%	50%	50%
FndNgt	3%	62%	8%
FntNgtR	2%	50%	0%
FldSum	3%	100%	0%
FldSumR	0%	0%	33%
ForSum	1%	50%	0%
ForSumR	2%	12%	75%
GrpRem	5%	85%	0%
MapOne	7%	76%	14%
[illegible]			
STMMap	3%	100%	0%
Scan	1%	43%	57%
SrtRDD	2%	70%	30%
StdDev	3%	25%	44%
StrCnt	2%	50%	50%
StrDem	2%	50%	0%
StrPer	4%	93%	0%

R - versions with changes

D - manually confirmed

I - invalid situations

Take Away So Far ...

Actual measurement protocol appears more important than subsequent computation

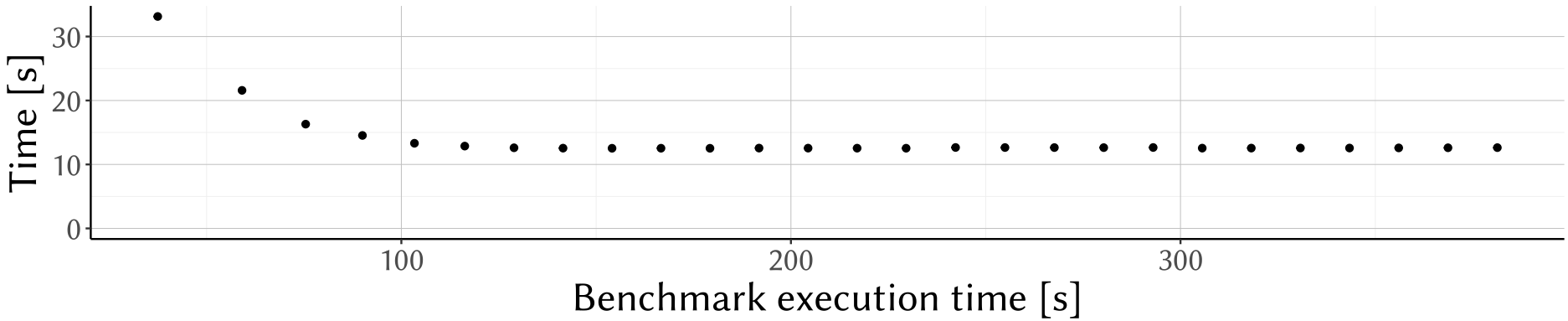
- Properly handling warm up
- Executing enough measurements
- Collecting supporting information

Change detection reliability per se not an issue

- Change definition issues beyond math
- Some benchmarks may require special attention

Outline

Warm Up



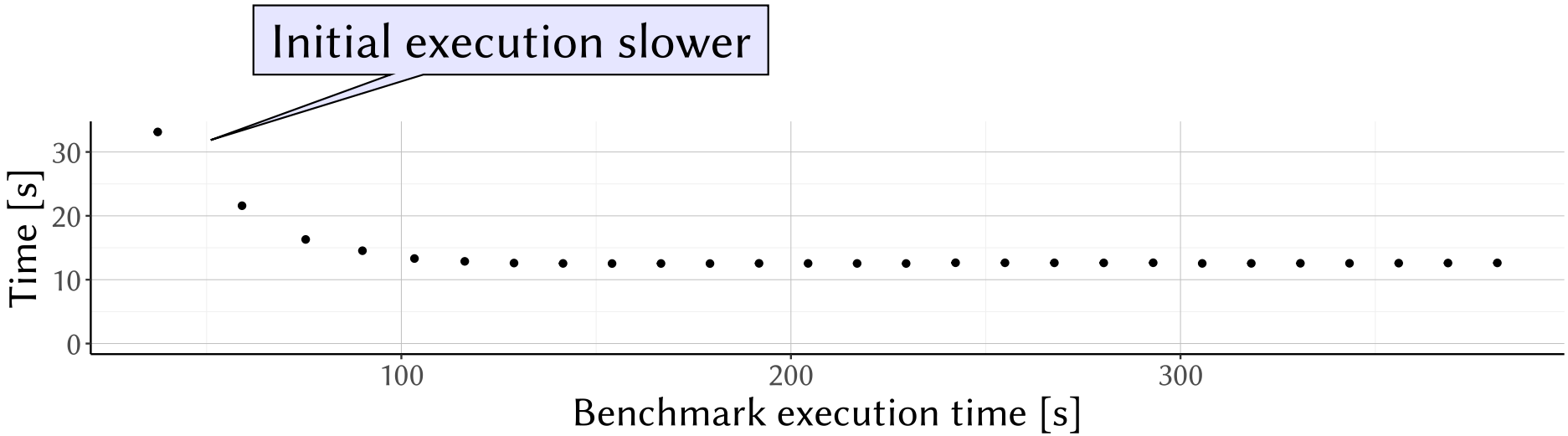
Plot Info

Input Benchmark repetition times for an arbitrarily selected benchmark and platform.

X axis Time from start of the benchmark execution.

Y axis Time of single benchmark repetition.

Warm Up



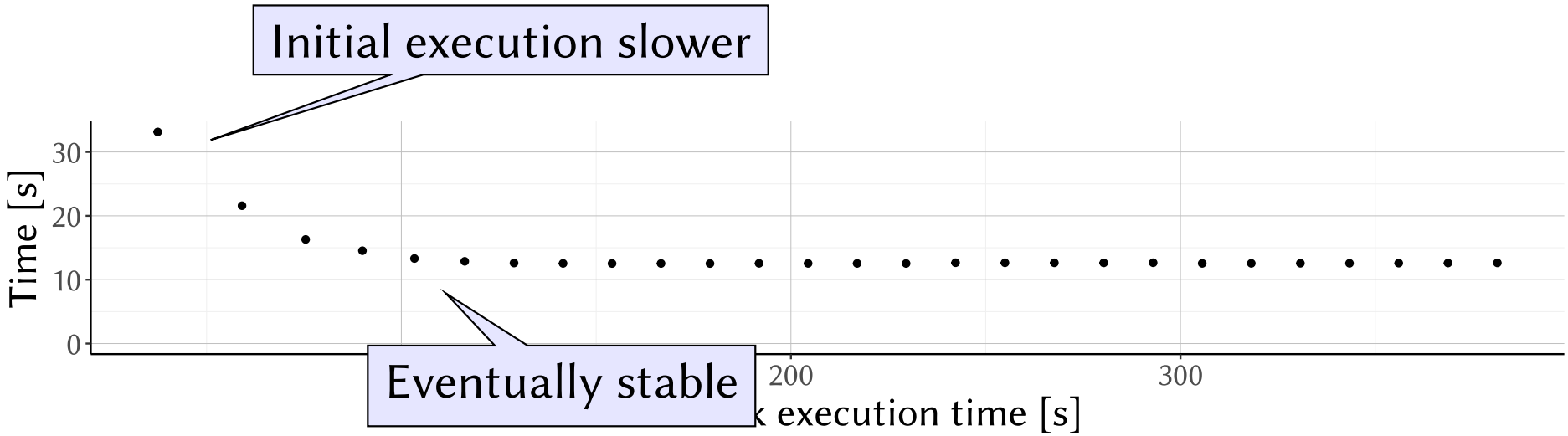
Plot Info

Input Benchmark repetition times for an arbitrarily selected benchmark and platform.

X axis Time from start of the benchmark execution.

Y axis Time of single benchmark repetition.

Warm Up



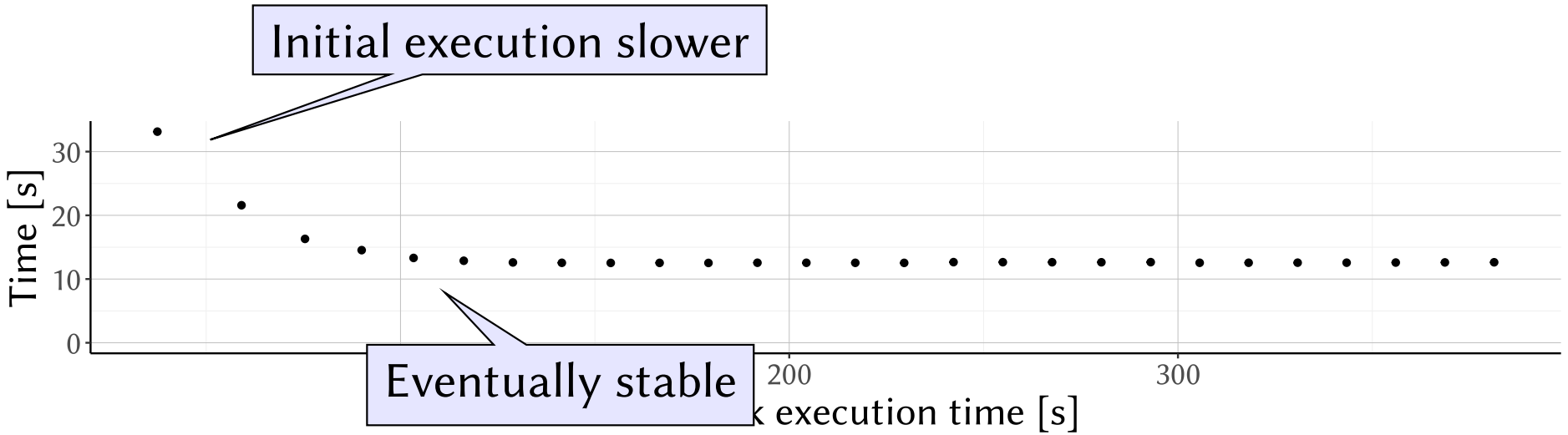
Plot Info

Input Benchmark repetition times for an arbitrarily selected benchmark and platform.

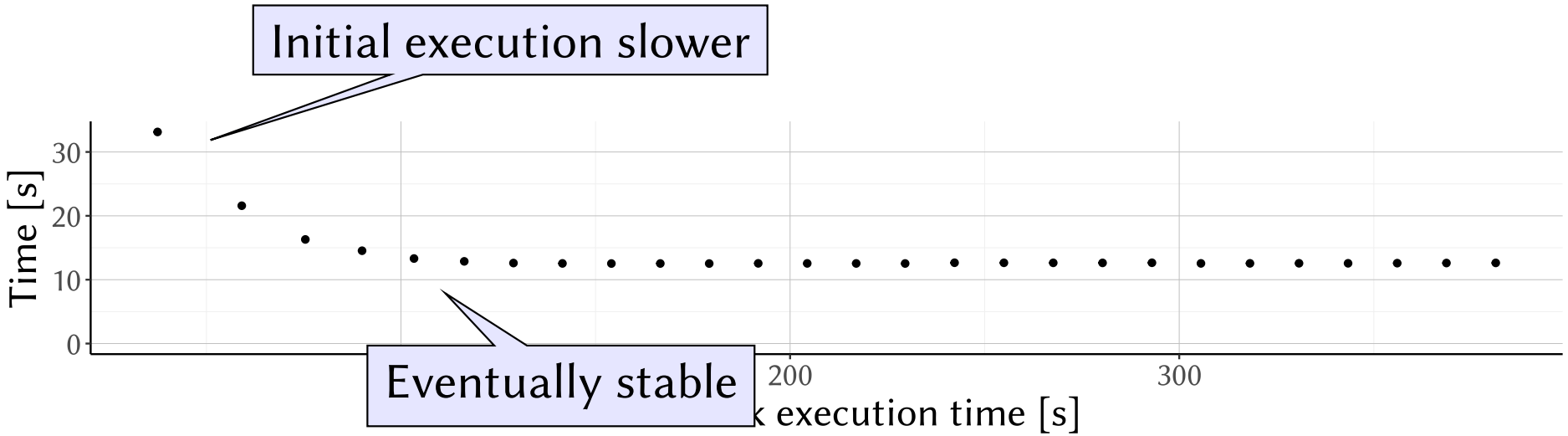
X axis Time from start of the benchmark execution.

Y axis Time of single benchmark repetition.

Warm Up


Initial execution slower
Eventually stable
Time [s]
0
10
20
30
200
300
execution time [s]

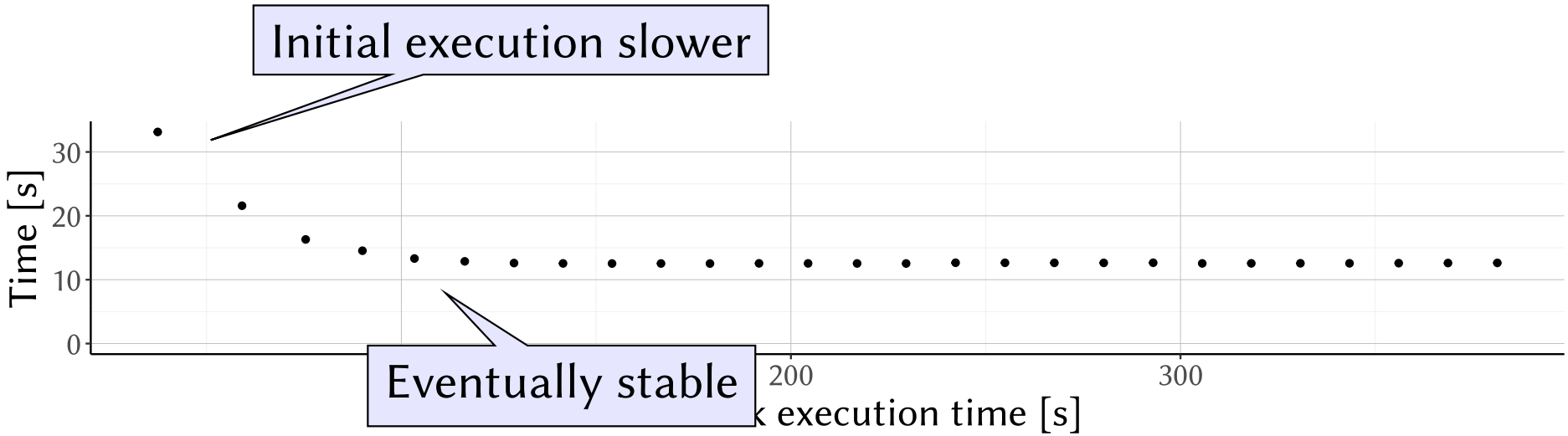
Warm Up



Some reasons behind warm up eliminated in our setup

- Most power management features disabled
- Initial and maximum heap size equal and fixed
- Most (but not all) benchmarks stable after first repetition

Warm Up

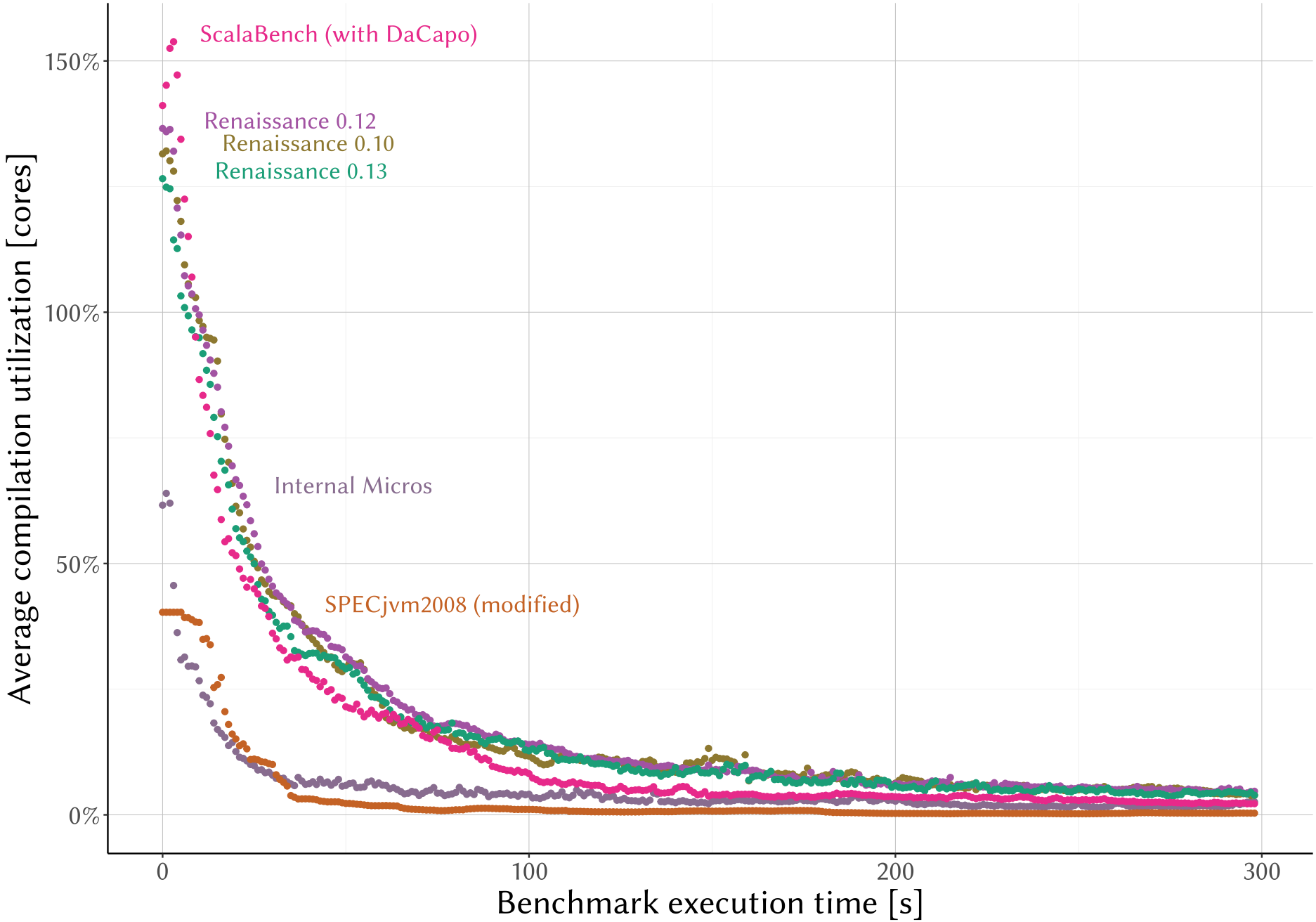


Some reasons behind warm up eliminated in our setup

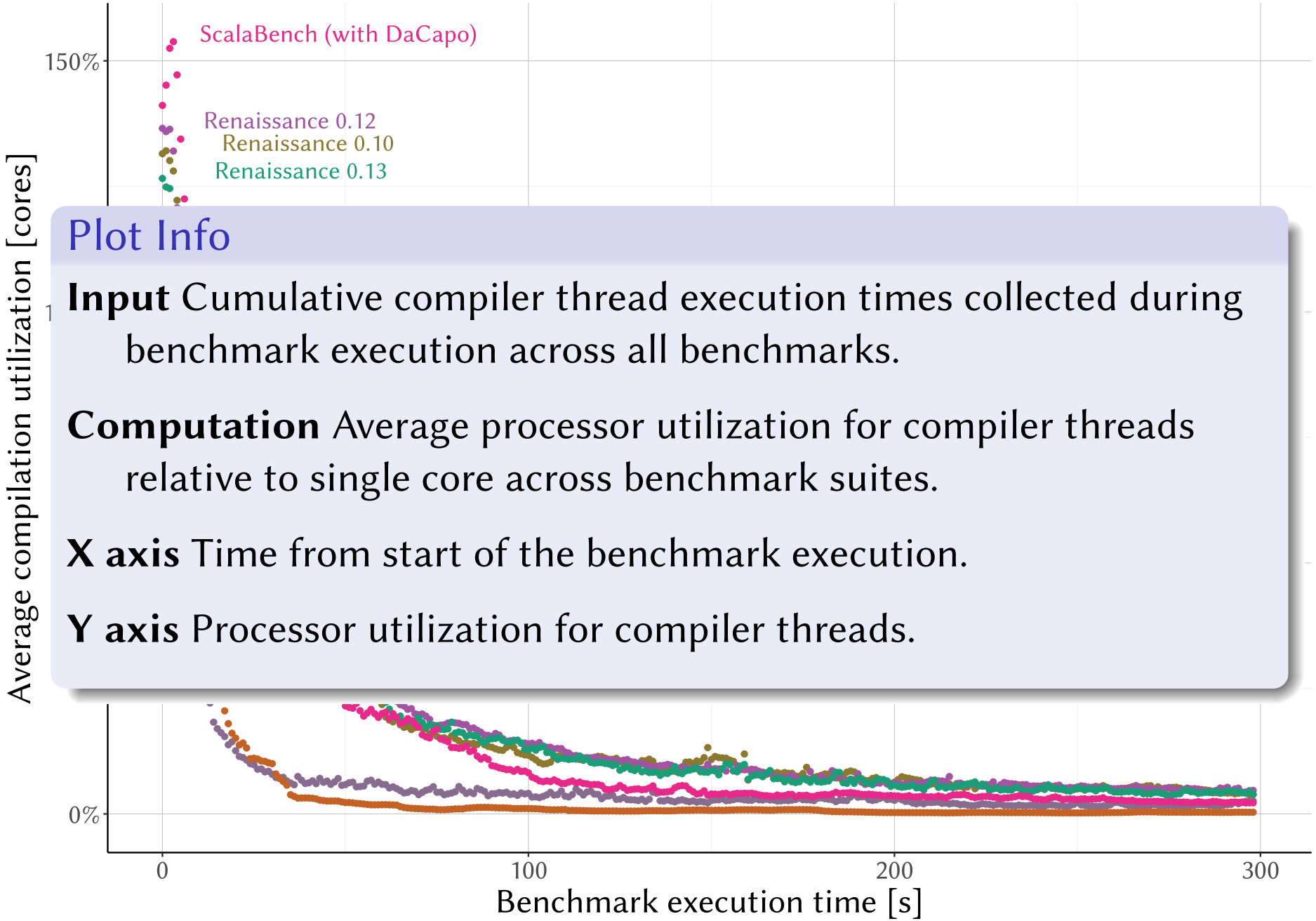
- Most power management features disabled
- Initial and maximum heap size equal and fixed
- Most (but not all) benchmarks stable after first repetition

But the elephant in the room is **just-in-time compilation**

How Much Compilation On Average ?



How Much Compilation On Average ?



Plot Info

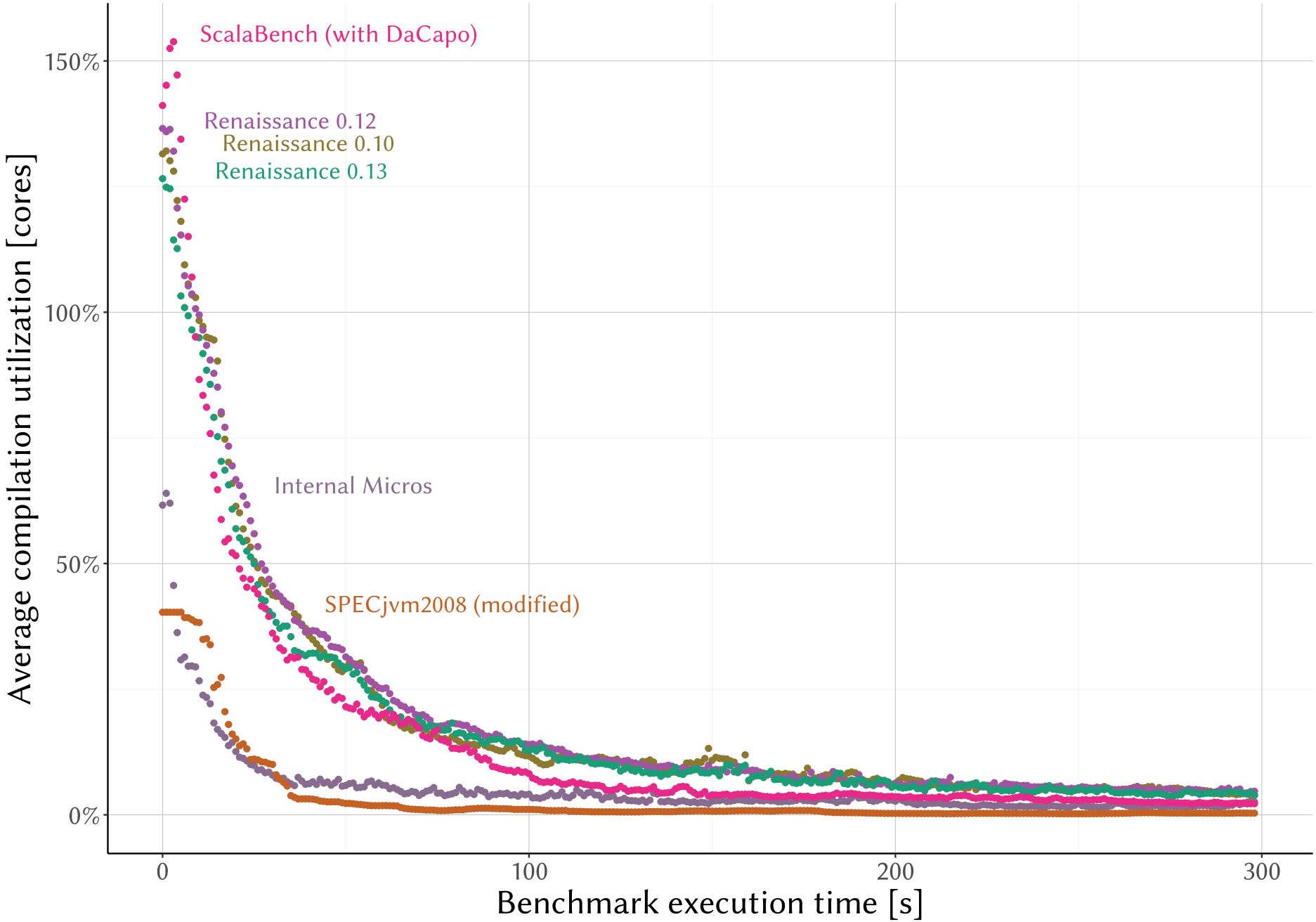
Input Cumulative compiler thread execution times collected during benchmark execution across all benchmarks.

Computation Average processor utilization for compiler threads relative to single core across benchmark suites.

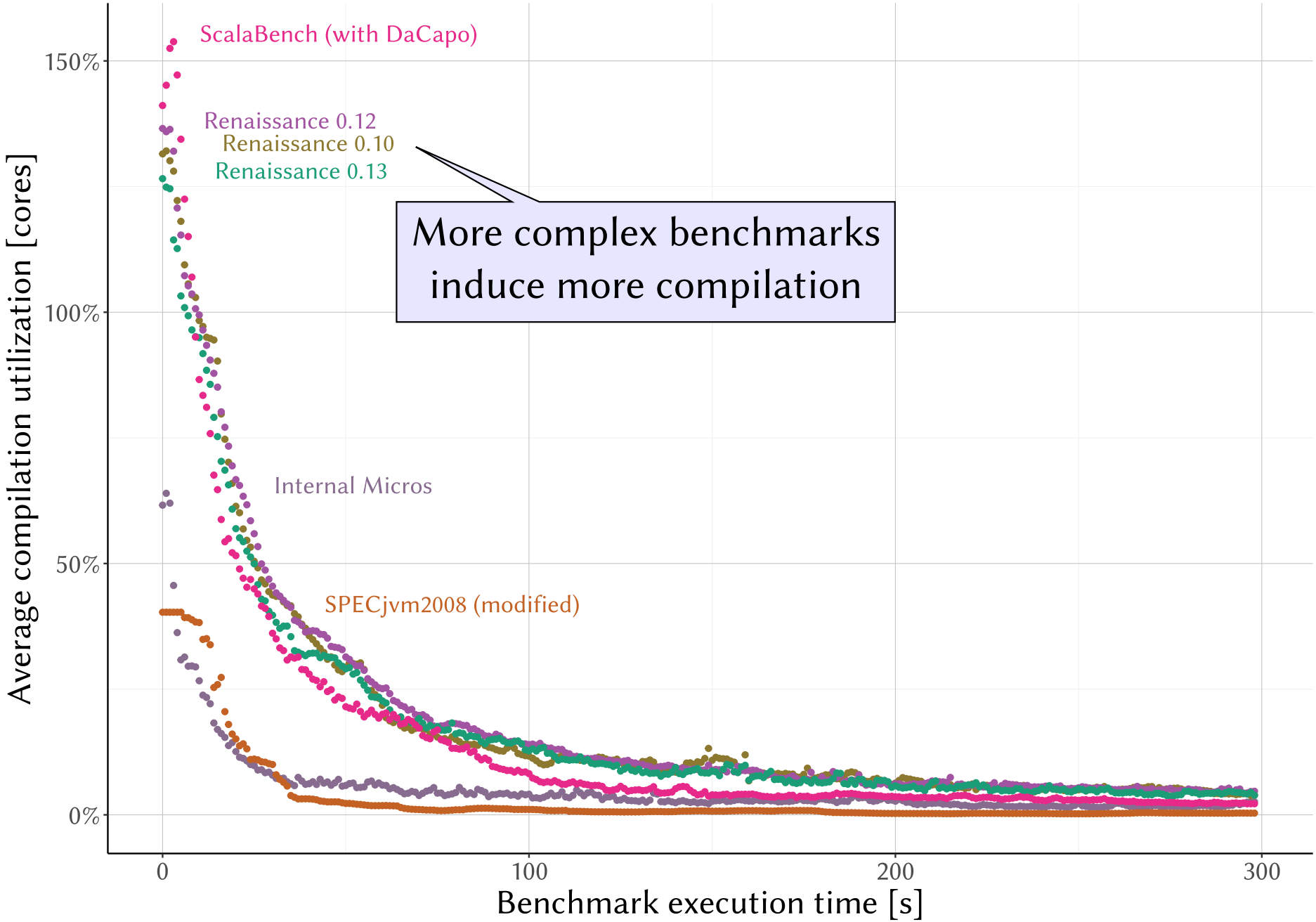
X axis Time from start of the benchmark execution.

Y axis Processor utilization for compiler threads.

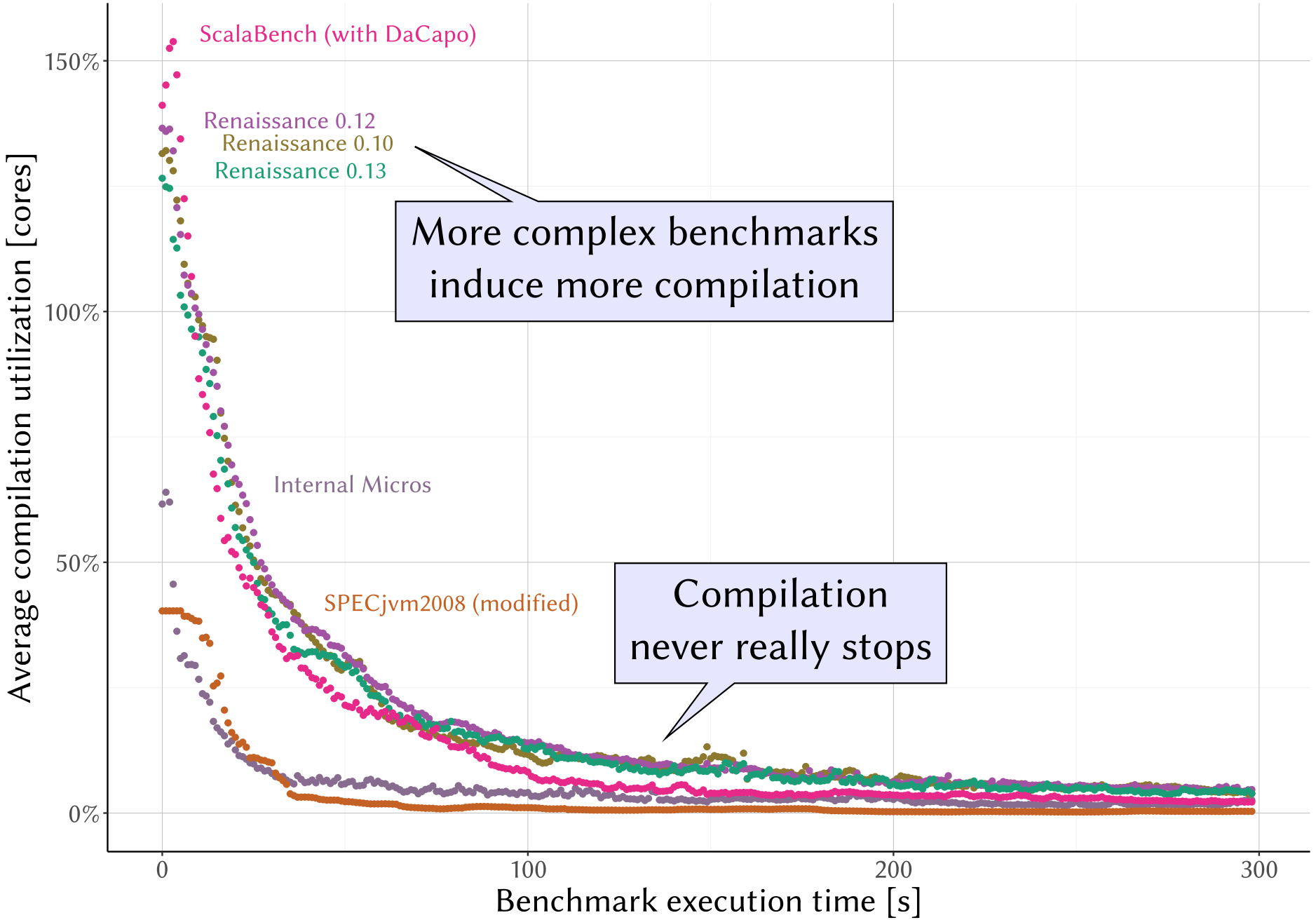
How Much Compilation On Average ?



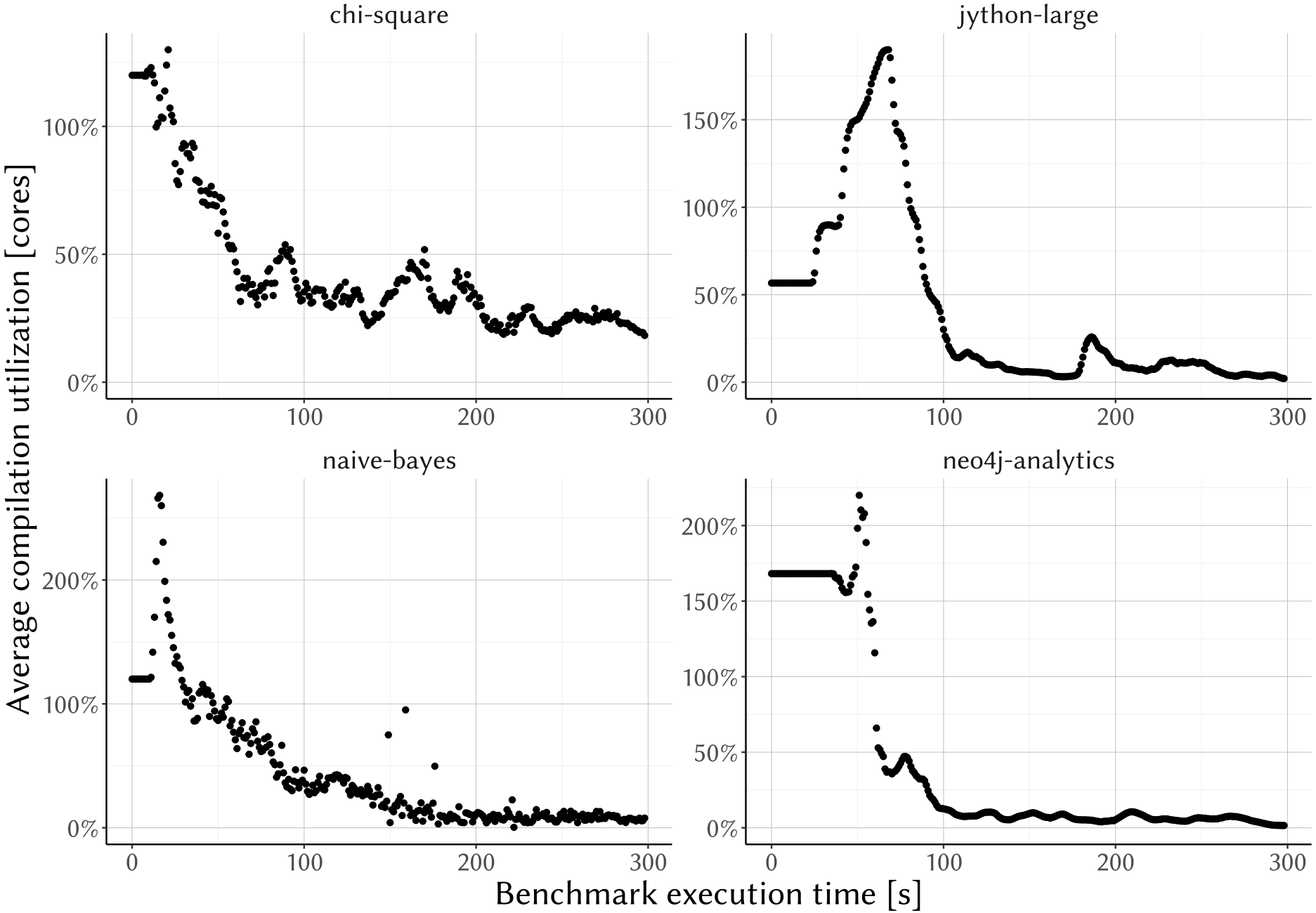
How Much Compilation On Average ?



How Much Compilation On Average ?



How Much Compilation Per Benchmark ?



Detecting Warm Up

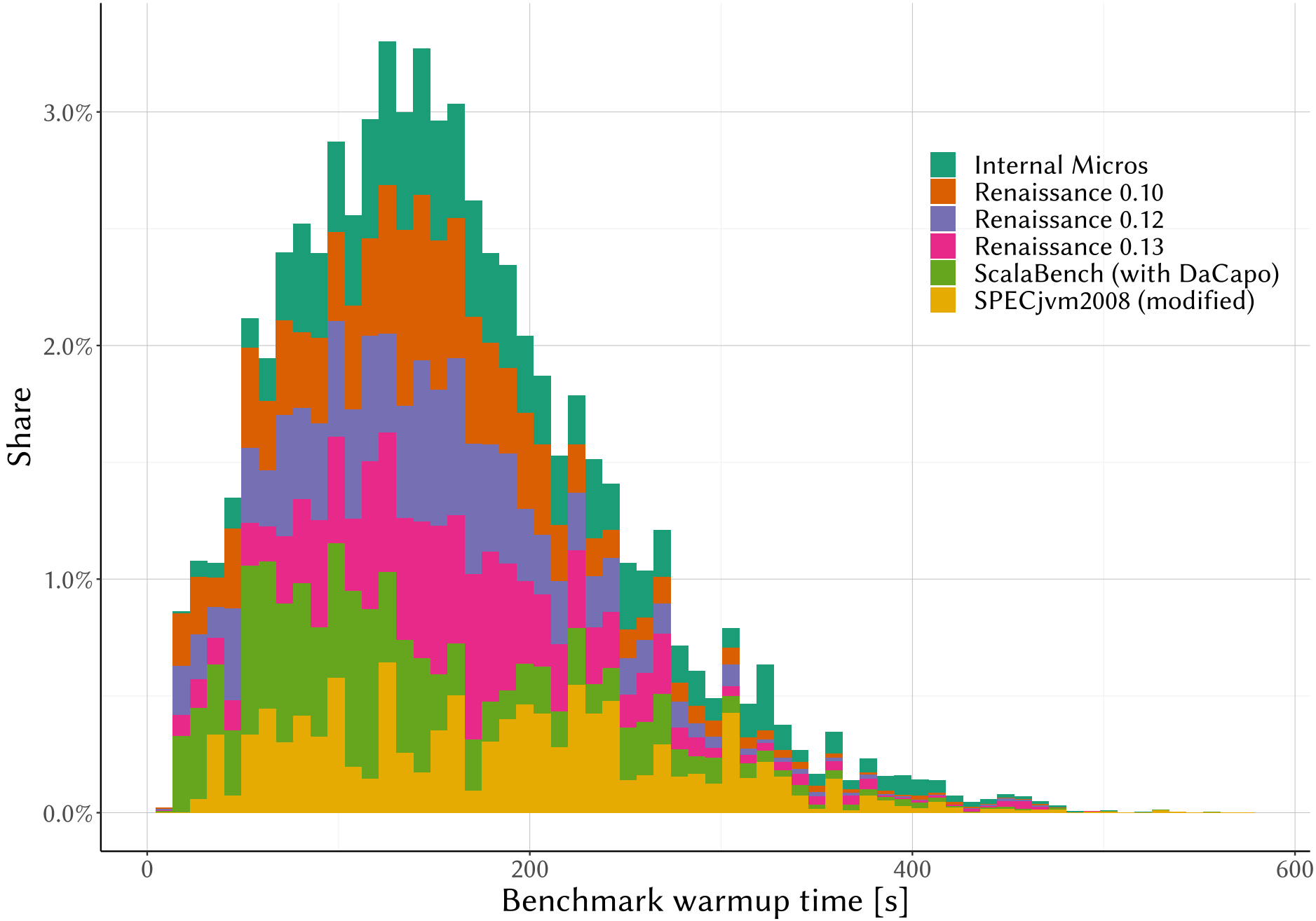
What do we want from warm up ?

- Make sure we measure code produced by the last tier compiler
- Move past the most egregious performance changes
- Do not waste too much time on warm up

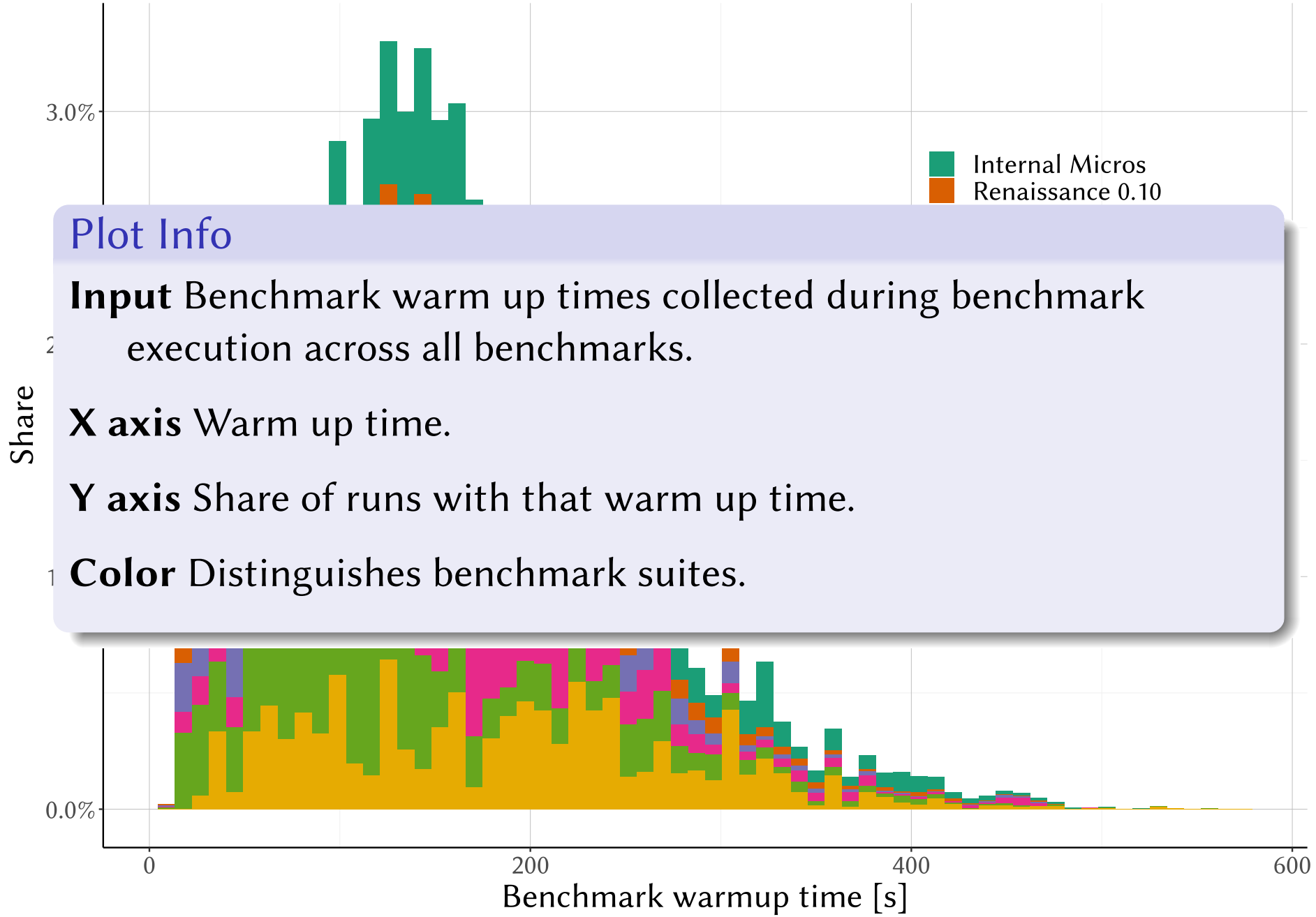
What we do

- Monitor activity of background compiler threads
- Establish thresholds across 60 s sliding window
- The first window with activity within 10 % of minimum is warm
  - The algorithm is not online
  - Used with runs of 300 s to 600 s
  - Will always identify some repetitions as warm

How Long Do We Warm Up ?



How Long Do We Warm Up ?



Plot Info

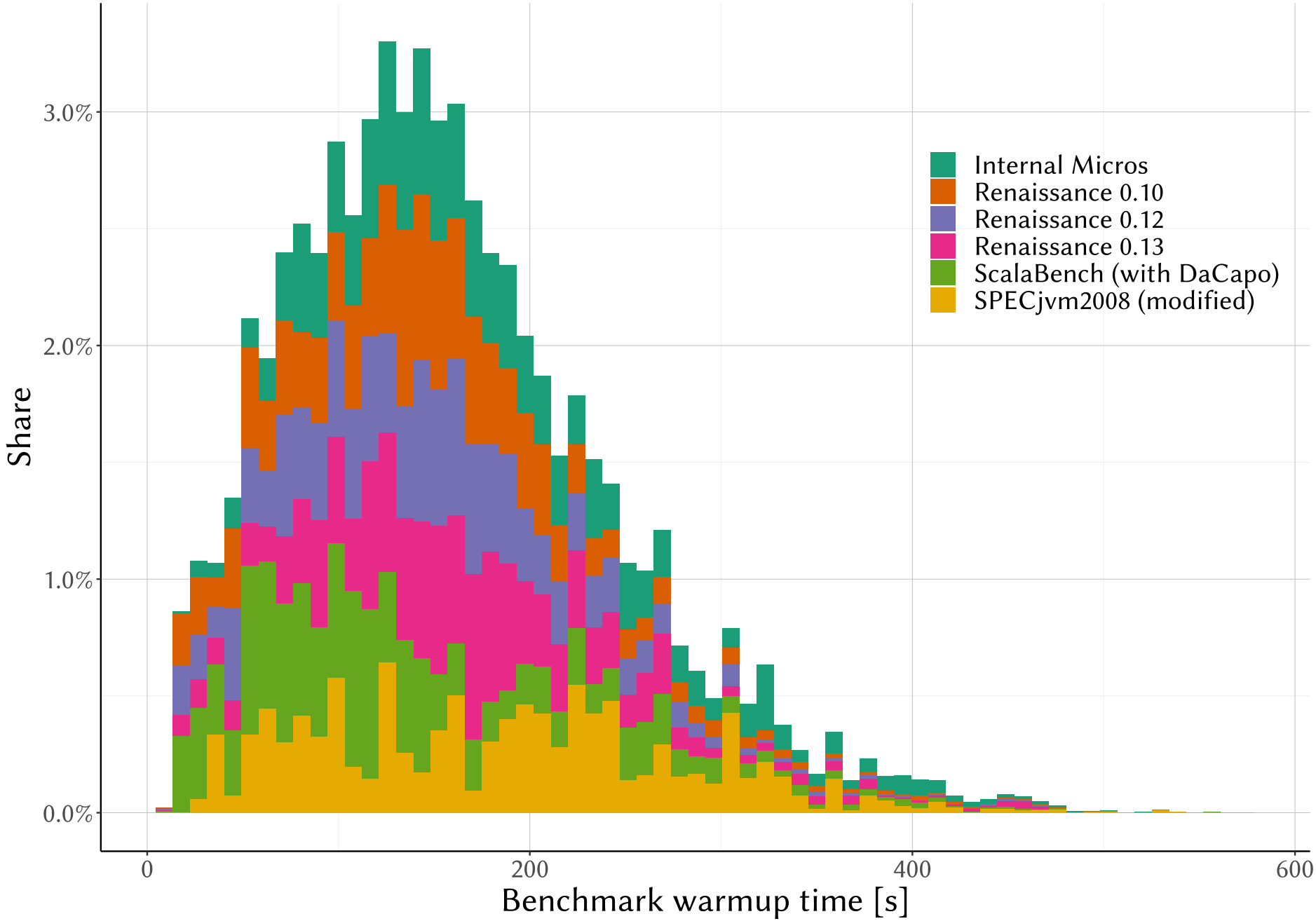
Input Benchmark warm up times collected during benchmark execution across all benchmarks.

X axis Warm up time.

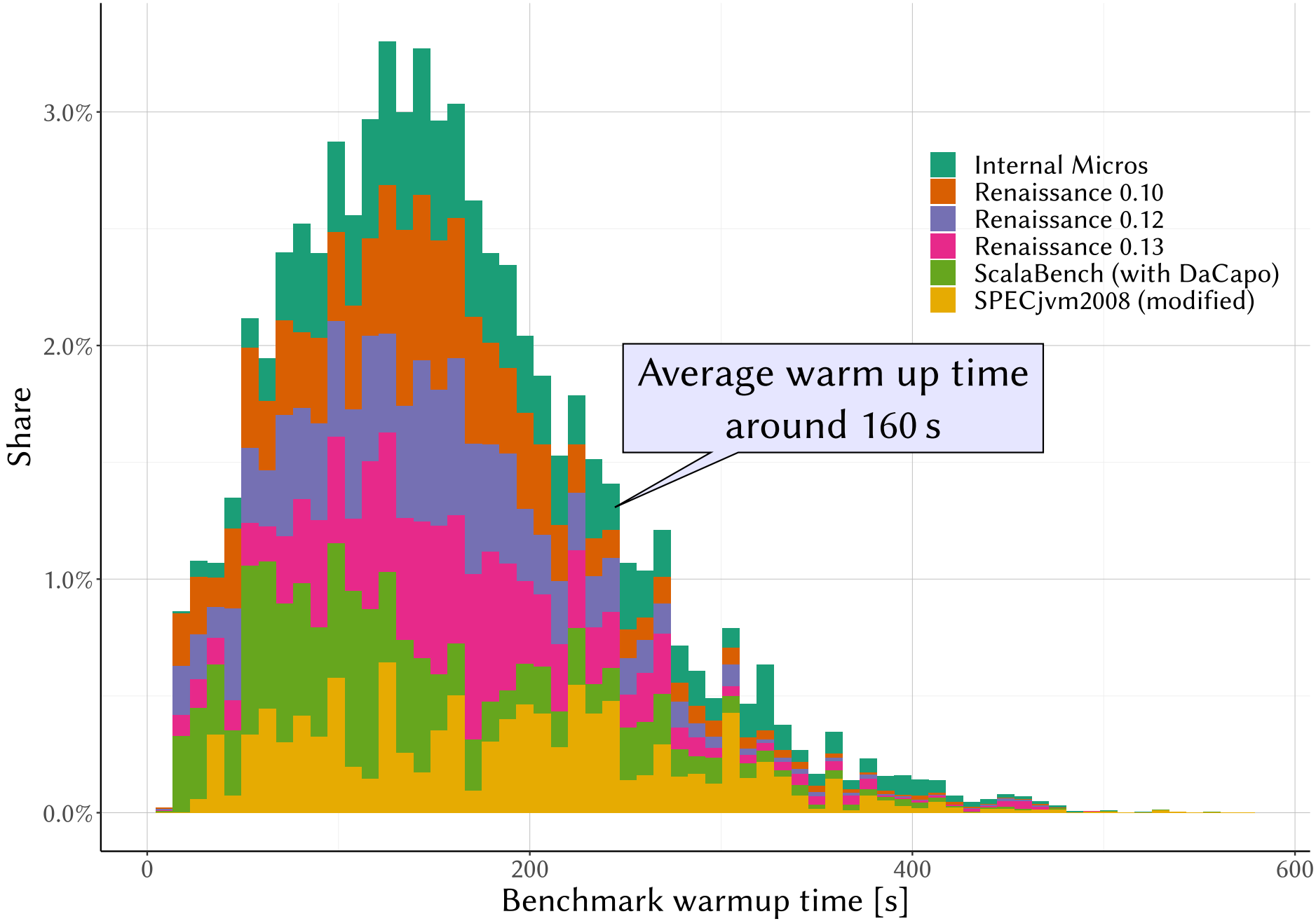
Y axis Share of runs with that warm up time.

Color Distinguishes benchmark suites.

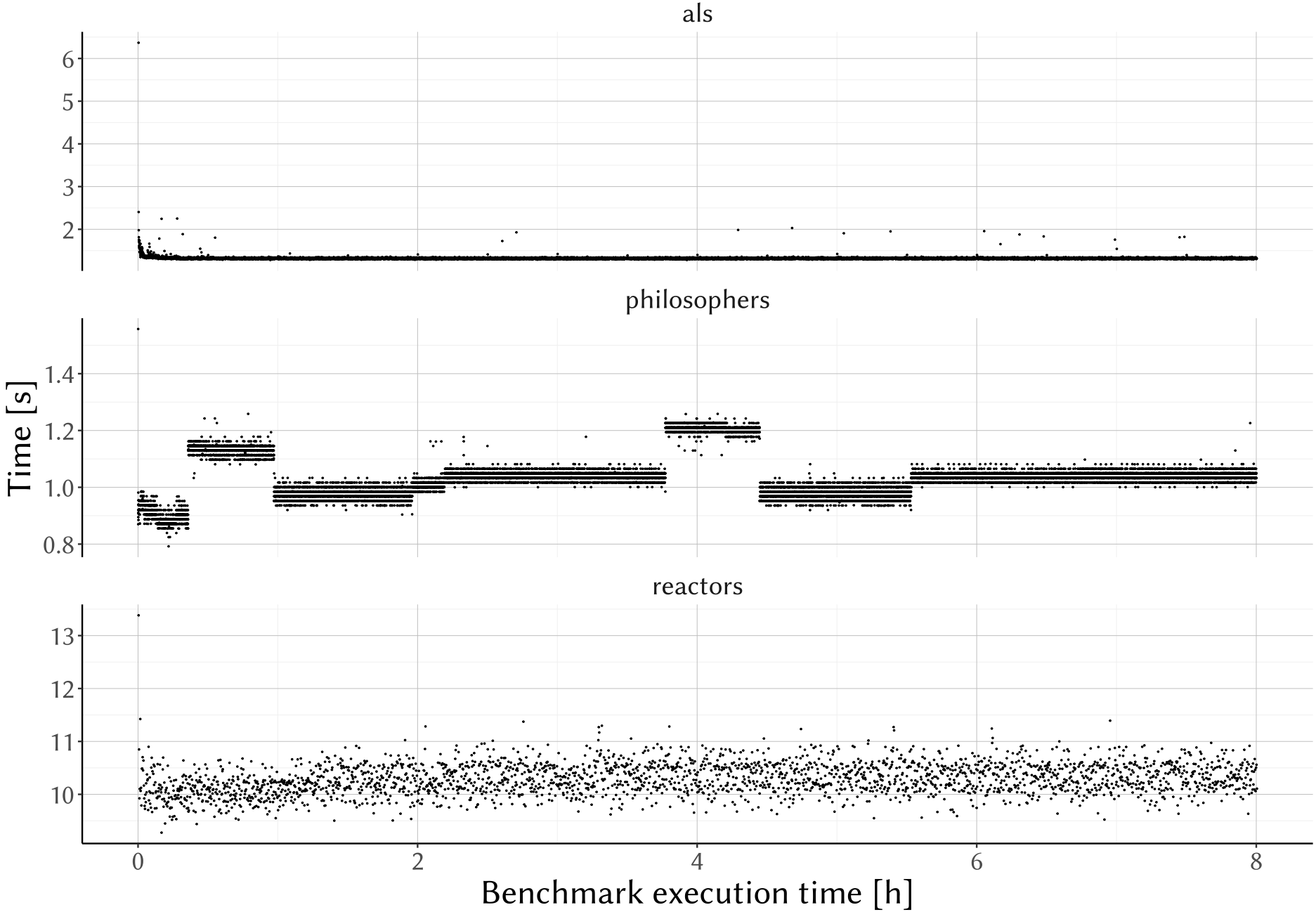
How Long Do We Warm Up ?



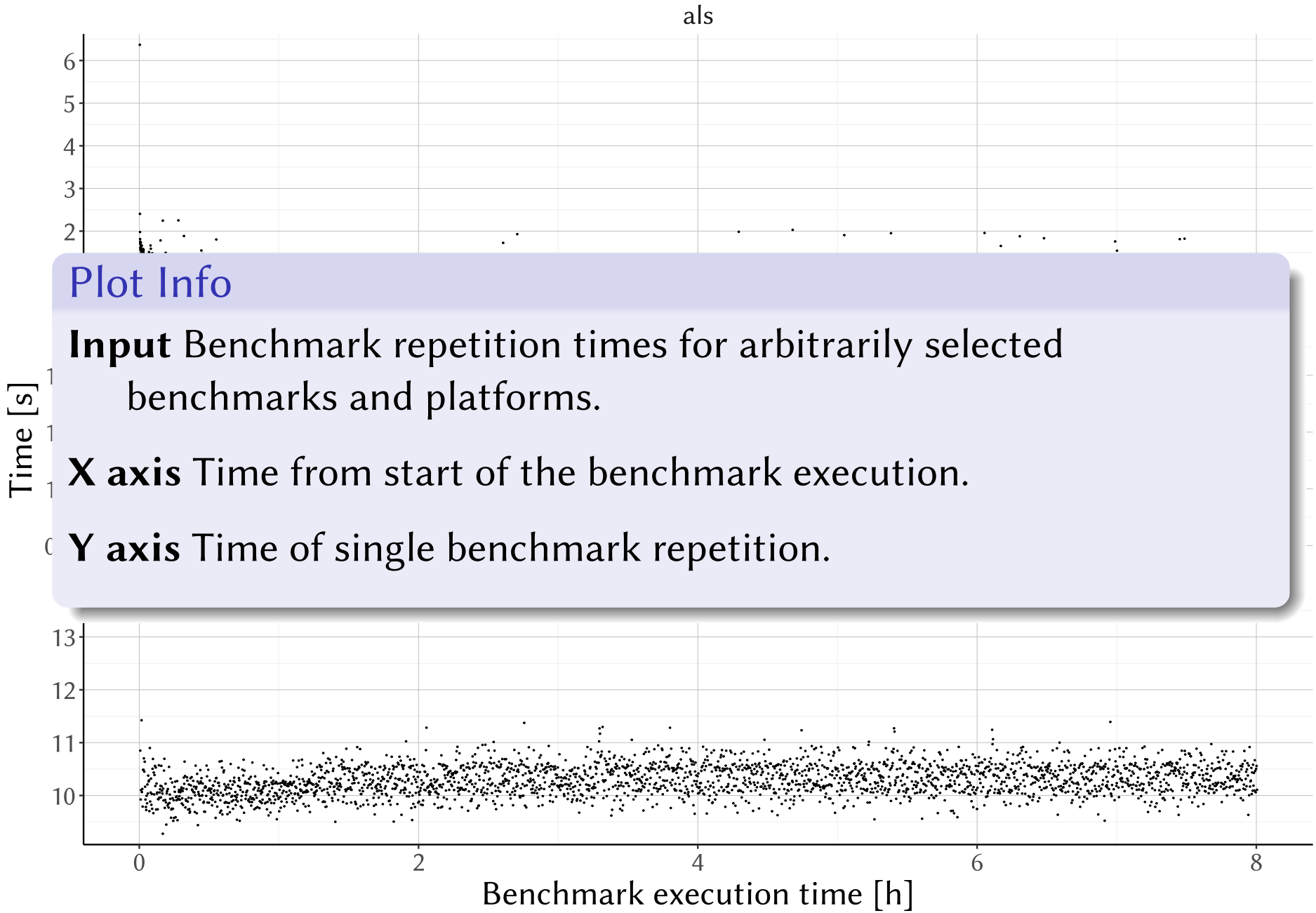
How Long Do We Warm Up ?



What About (Much) Longer Warm Up ?



What About (Much) Longer Warm Up ?



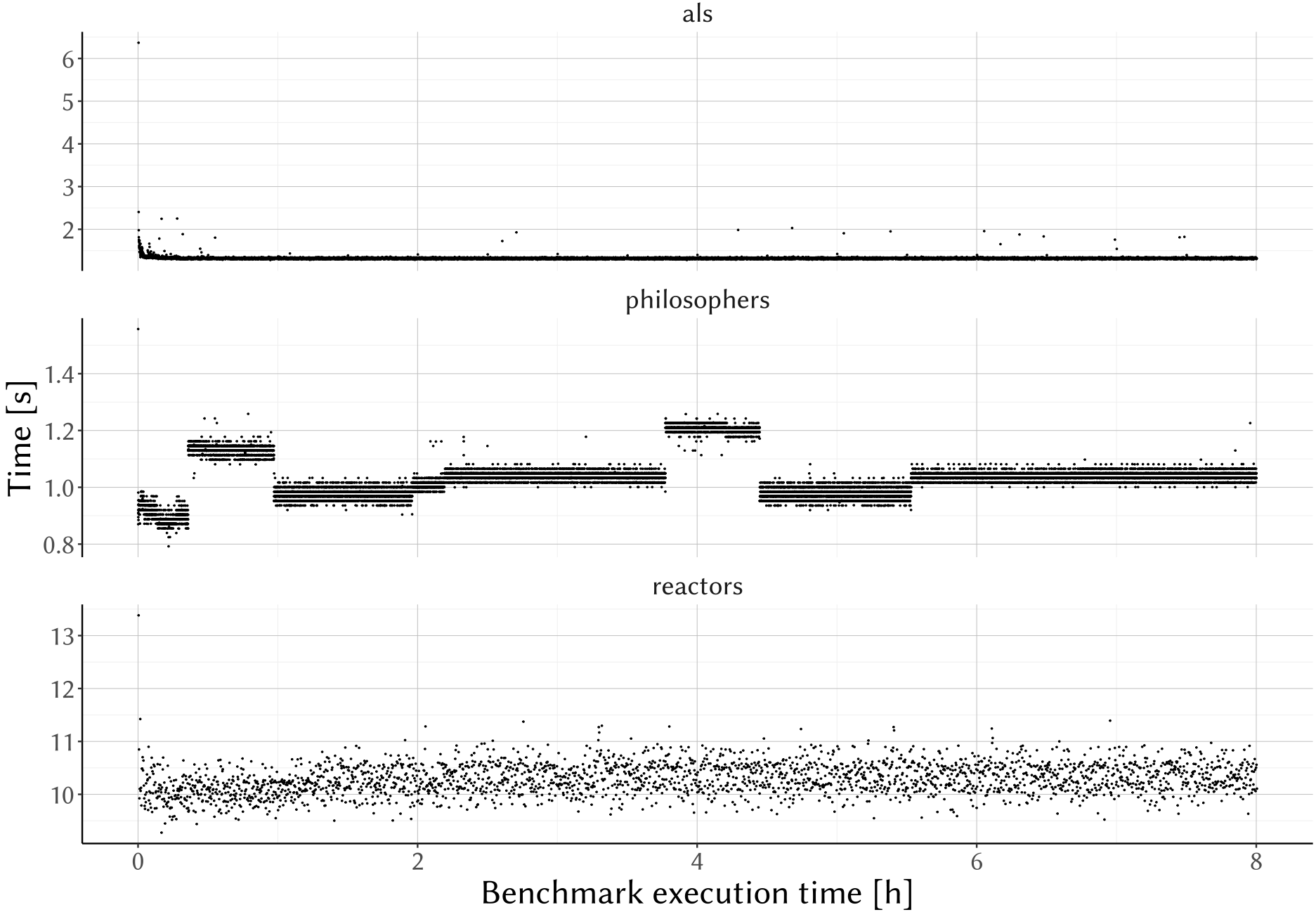
Plot Info

Input Benchmark repetition times for arbitrarily selected benchmarks and platforms.

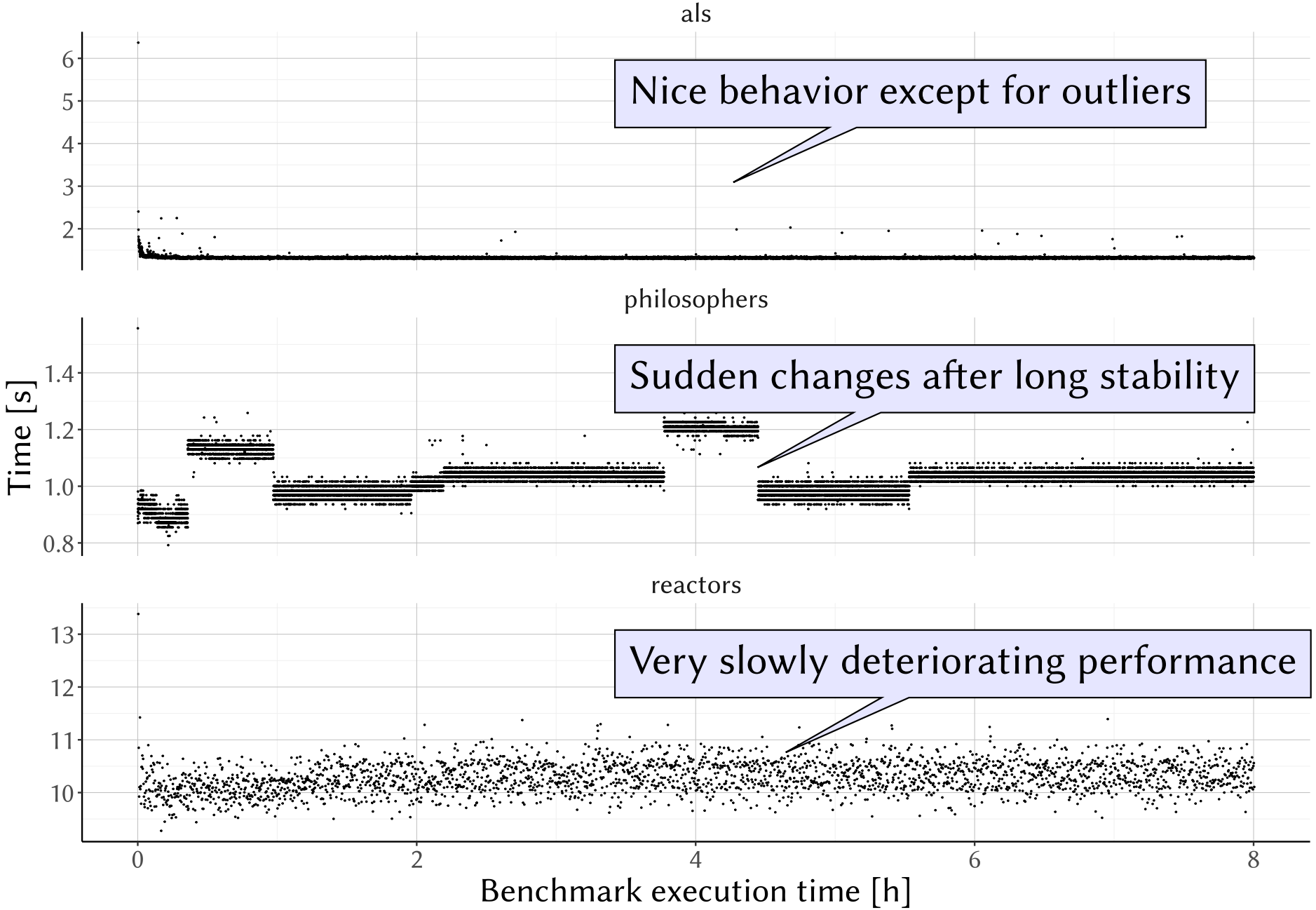
X axis Time from start of the benchmark execution.

Y axis Time of single benchmark repetition.

What About (Much) Longer Warm Up ?



What About (Much) Longer Warm Up ?



Take Away So Far ...

Some warm up properties complicate detection from time measurements

- Performance can change at any time into benchmark execution
- Performance changes possibly rather sudden
- Performance changes in both directions

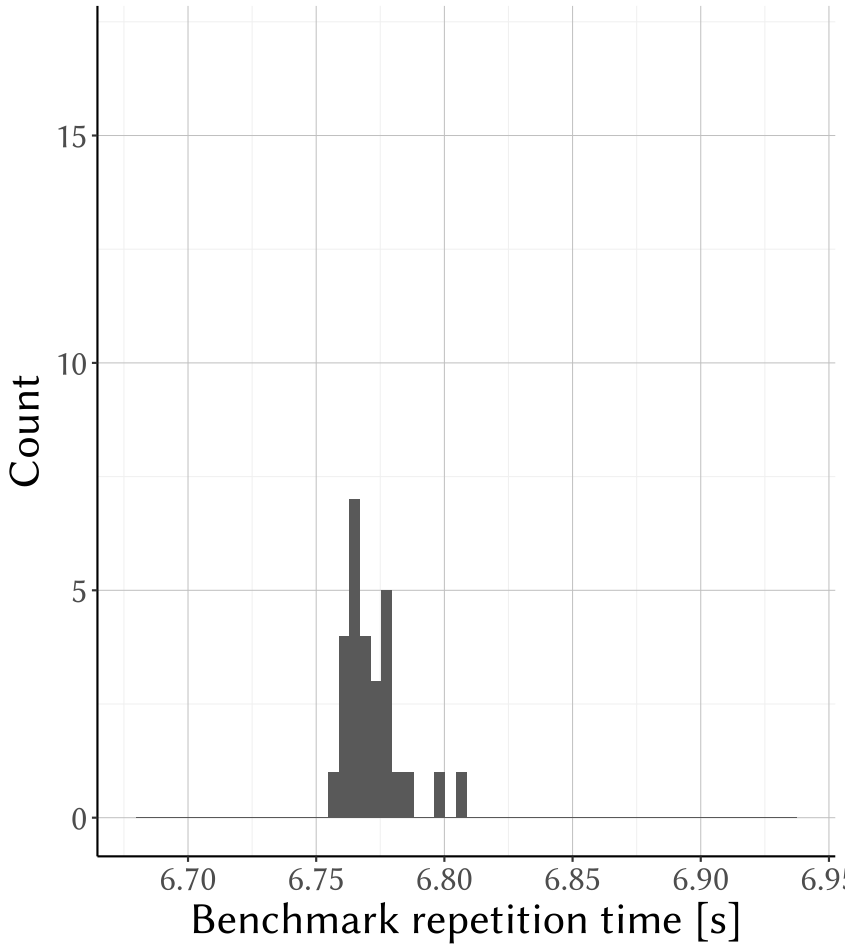
Reaching measurement stability not really the goal here

- Looking (only) at repetition times possibly wrong
- Warm up detection surprisingly important
  - Too much warmup is prohibitive resource hog
  - Too little warmup produces useless measurements

Outline

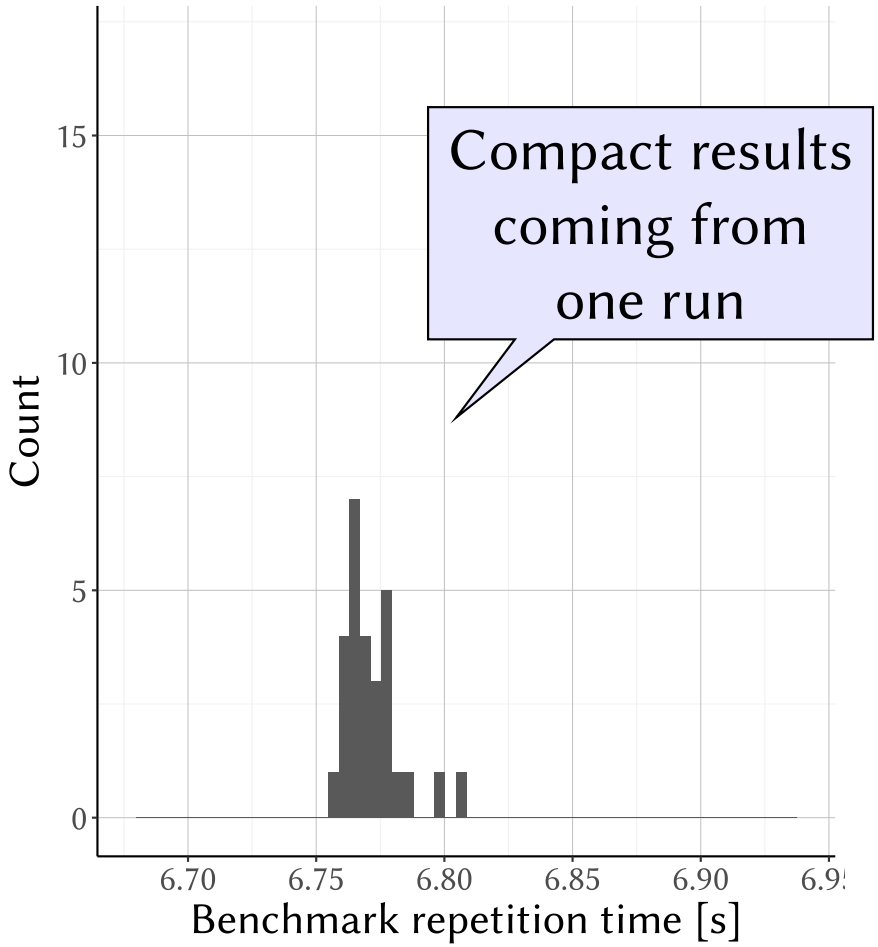
Handling More Runs

A single benchmark run does not really tell the whole story ...



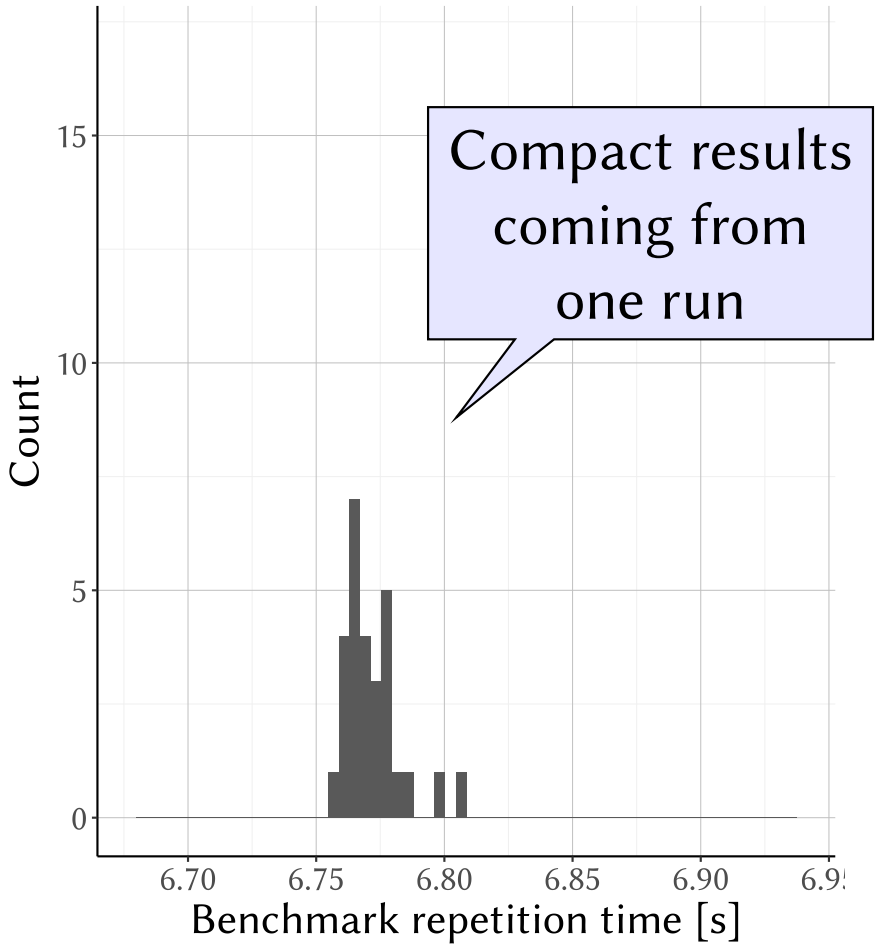
Handling More Runs

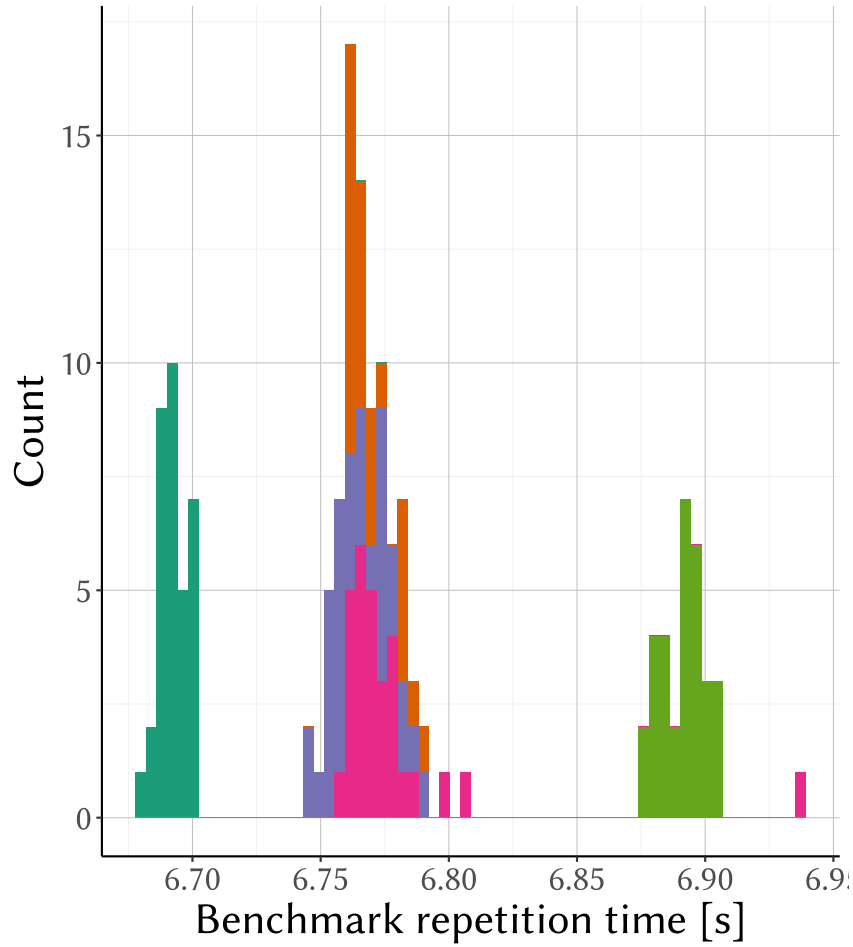
A single benchmark run does not really tell the whole story ...



Handling More Runs

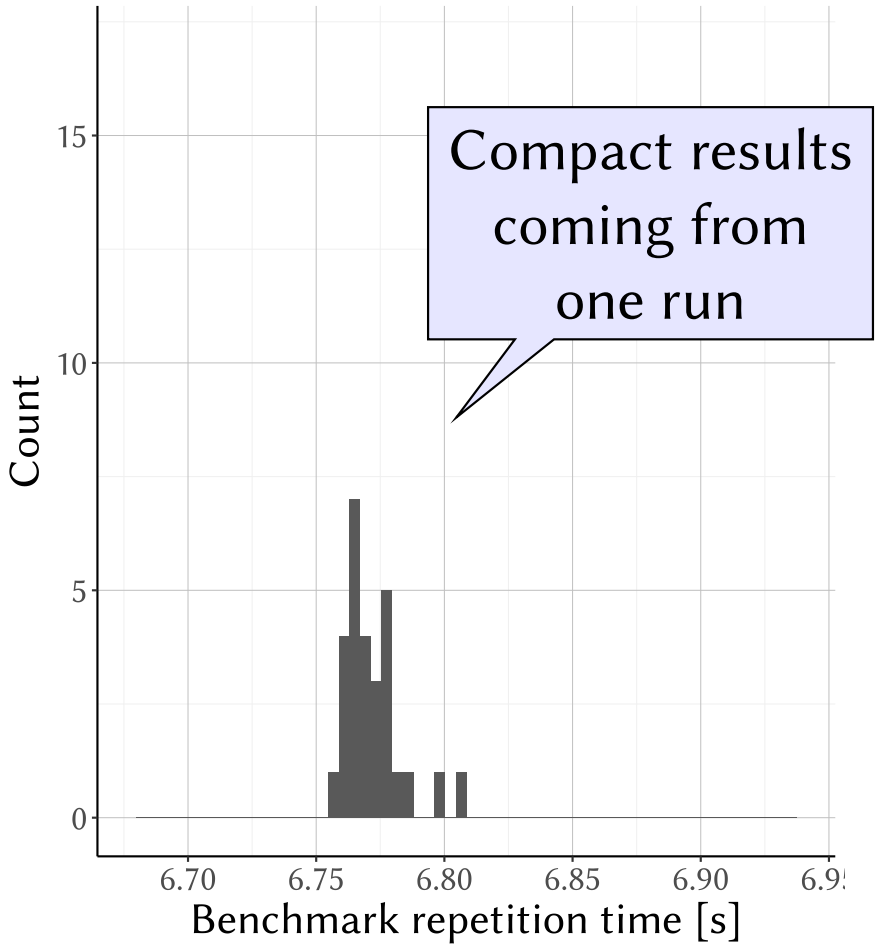
A single benchmark run does not really tell the whole story ...

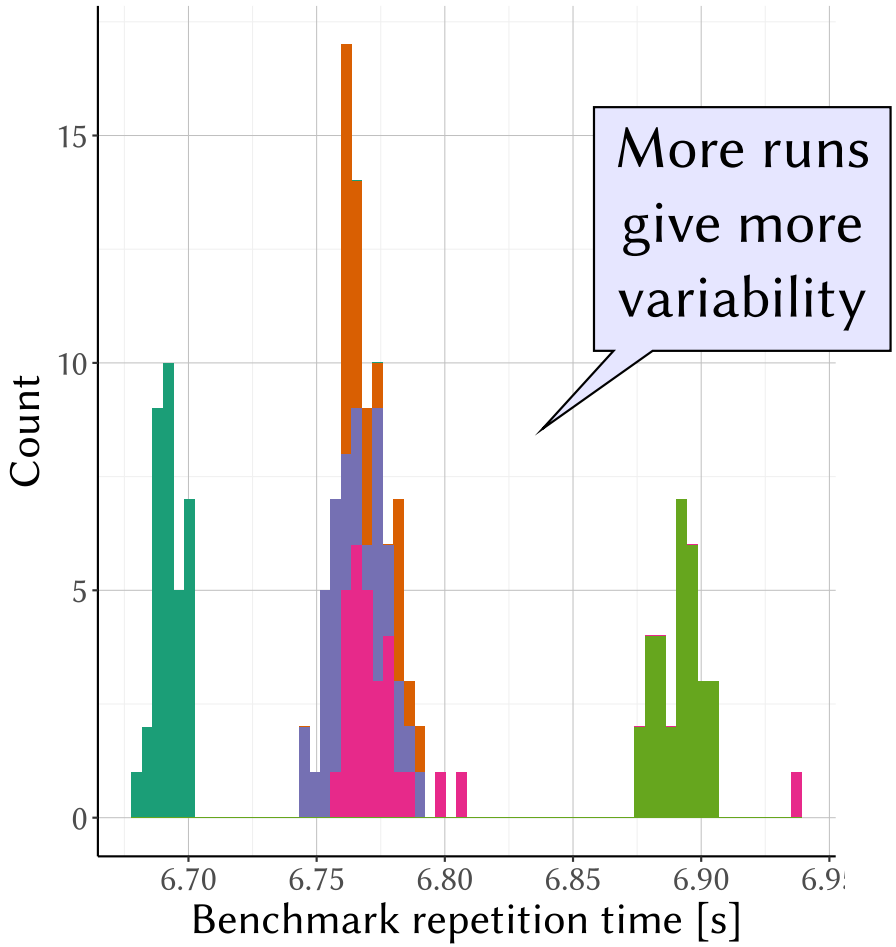




Handling More Runs

A single benchmark run does not really tell the whole story ...





How Many Runs Needed ...

... to compute average performance with at most 1 % error in 99 % of cases ?

Renaissance 0.10				
bench	C8	C11	E8	E11
aka-uct	99+	40	68	99+
als	99+	7	73	99+
chi-sqr	99+	99+		99+
db-shot	99+	99+	47	60
dec-tre	73	94	31	24
dotty	17	13	14	10
fin-chi	99+	99+	99+	99+
fin-htt	58	21	53	22
fj-kmns	12	6	72	16
fut-gen	99+	99+	99+	99+
gauss	99+	99+	99+	99+
log-reg	34	35	26	19
mne	99+	99+	62	99+
mov-len	6	19	6	14
nai-bay	5	5	99+	99+
neo-ana	99+	94	99+	99+
pg-rank	99+	99+	99+	46
par-mne	78	84	53	99+
philos	99+	99+	99+	99+
reactr	99+	53	99+	99+

rx-scrb	75	60	20	23
sc-doku	99+	99+	99+	99+
sc-kmns	53	38	9	10
sc-stmb	71	33	61	99+
scrb	99+	99+	99+	99+
ScalaBench (with DaCapo)				
bench	C8	C11	E8	E11
	59		19	
appar-d	99+	99+	63	99+
avror-l	5	11	8	2
batik-s	1	4	2	7
eclps-s	3		16	
factr-d	99+	99+	99+	99+
fop-d	18	23	20	15
h2-d	27	37	29	98
jythn-l	31	99+	21	56
kiama-d	37	48	42	16
luidx-d	94	64	30	34
lusrc-l	38	34	27	18
pmd-l	44	59	24	18
scc-l	65	99+	21	22
scdoc-l	54	78	38	47

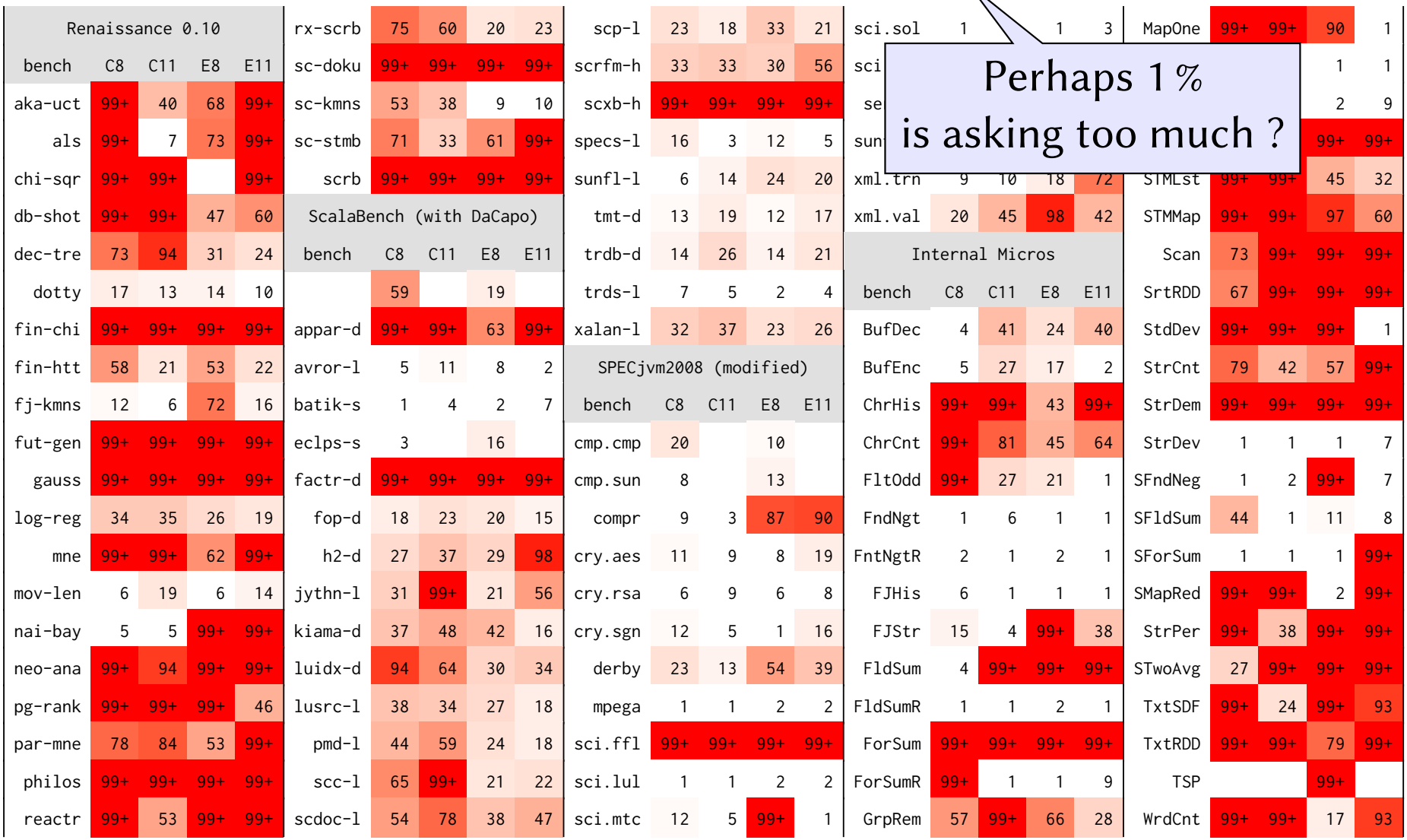
scp-l	23	18	33	21
scrfm-h	33	33	30	56
scxb-h	99+	99+	99+	99+
specs-l	16	3	12	5
sunfl-l	6	14	24	20
tmt-d	13	19	12	17
trdb-d	14	26	14	21
trds-l	7	5	2	4
xalan-l	32	37	23	26
SPECjvm2008 (modified)				
bench	C8	C11	E8	E11
cmp.cmp	20		10	
cmp.sun	8		13	
compr	9	3	87	90
cry.aes	11	9	8	19
cry.rsa	6	9	6	8
cry.sgn	12	5	1	16
derby	23	13	54	39
mpega	1	1	2	2
sci.ffl	99+	99+	99+	99+
sci.lul	1	1	2	2
sci.mtc	12	5	99+	1

sci.sol	1	1	1	3
sci.spl	1	11	1	1
serial	12	10	32	99+
sunflow	7	6	7	7
xml.trn	9	10	18	72
xml.val	20	45	98	42
Internal Micros				
bench	C8	C11	E8	E11
BufDec	4	41	24	40
BufEnc	5	27	17	2
ChrHis	99+	99+	43	99+
ChrCnt	99+	81	45	64
FltOdd	99+	27	21	1
FndNgt	1	6	1	1
FntNgtR	2	1	2	1
FJHis	6	1	1	1
FJStr	15	4	99+	38
FldSum	4	99+	99+	99+
FldSumR	1	1	2	1
ForSum	99+	99+	99+	99+
ForSumR	99+	1	1	9
GrpRem	57	99+	66	28

MapOne	99+	99+	90	1
NetDot	1	1	1	1
NetEig	1	2	2	9
Reduce	99+	99+	99+	99+
STMLst	99+	99+	45	32
STMMap	99+	99+	97	60
Scan	73	99+	99+	99+
SrtRDD	67	99+	99+	99+
StdDev	99+	99+	99+	1
StrCnt	79	42	57	99+
StrDem	99+	99+	99+	99+
StrDev	1	1	1	7
SFndNeg	1	2	99+	7
SFldSum	44	1	11	8
SForSum	1	1	1	99+
SMapRed	99+	99+	2	99+
StrPer	99+	38	99+	99+
STwoAvg	27	99+	99+	99+
TxtSDF	99+	24	99+	93
TxtRDD	99+	99+	79	99+
TSP			99+	
WrdCnt	99+	99+	17	93

How Many Runs Needed ...

... to compute average performance with at most 1 % error in 99 % of cases ?



Renaissance 0.10				
bench	C8	C11	E8	E11
aka-uct	99+	40	68	99+
als	99+	7	73	99+
chi-sqr	99+	99+		99+
db-shot	99+	99+	47	60
dec-tre	73	94	31	24
dotty	17	13	14	10
fin-chi	99+	99+	99+	99+
fin-htt	58	21	53	22
fj-kmns	12	6	72	16
fut-gen	99+	99+	99+	99+
gauss	99+	99+	99+	99+
log-reg	34	35	26	19
mne	99+	99+	62	99+
mov-len	6	19	6	14
nai-bay	5	5	99+	99+
neo-ana	99+	94	99+	99+
pg-rank	99+	99+	99+	46
par-mne	78	84	53	99+
philos	99+	99+	99+	99+
reactr	99+	53	99+	99+

rx-scrb	75	60	20	23
sc-doku	99+	99+	99+	99+
sc-kmns	53	38	9	10
sc-stmb	71	33	61	99+
scrb	99+	99+	99+	99+

ScalaBench (with DaCapo)				
bench	C8	C11	E8	E11
	59		19	
appar-d	99+	99+	63	99+
avror-l	5	11	8	2
batik-s	1	4	2	7
eclps-s	3		16	
factr-d	99+	99+	99+	99+
fop-d	18	23	20	15
h2-d	27	37	29	98
jythn-l	31	99+	21	56
kiama-d	37	48	42	16
luidx-d	94	64	30	34
lusrc-l	38	34	27	18
pmd-l	44	59	24	18
scc-l	65	99+	21	22
scdoc-l	54	78	38	47

scp-l	23	18	33	21
scrfm-h	33	33	30	56
scxb-h	99+	99+	99+	99+
specs-l	16	3	12	5
sunfl-l	6	14	24	20
tmt-d	13	19	12	17
trdb-d	14	26	14	21
trds-l	7	5	2	4
xalan-l	32	37	23	26

SPECjvm2008 (modified)				
bench	C8	C11	E8	E11
cmp.cmp	20		10	
cmp.sun	8		13	
compr	9	3	87	90
cry.aes	11	9	8	19
cry.rsa	6	9	6	8
cry.sgn	12	5	1	16
derby	23	13	54	39
mpega	1	1	2	2
sci.ffl	99+	99+	99+	99+
sci.lul	1	1	2	2
sci.mtc	12	5	99+	1

sci.sol	1		1	3
sci				1
se				2
sun				99+
xml.trn	9	10	18	72
xml.val	20	45	98	42

Internal Micros				
bench	C8	C11	E8	E11
BufDec	4	41	24	40
BufEnc	5	27	17	2
ChrHis	99+	99+	43	99+
ChrCnt	99+	81	45	64
FltOdd	99+	27	21	1
FndNgt	1	6	1	1
FntNgtR	2	1	2	1
FJHis	6	1	1	1
FJStr	15	4	99+	38
FldSum	4	99+	99+	99+
FldSumR	1	1	2	1
ForSum	99+	99+	99+	99+
ForSumR	99+	1	1	9
GrpRem	57	99+	66	28

MapOne	99+	99+	90	1
STMLst	99+	99+	45	32
STMMap	99+	99+	97	60
Scan	73	99+	99+	99+
SrtRDD	67	99+	99+	99+
StdDev	99+	99+	99+	1
StrCnt	79	42	57	99+
StrDem	99+	99+	99+	99+
StrDev	1	1	1	7
SFndNeg	1	2	99+	7
SFldSum	44	1	11	8
SForSum	1	1	1	99+
SMapRed	99+	99+	2	99+
StrPer	99+	38	99+	99+
STwoAvg	27	99+	99+	99+
TxtSDF	99+	24	99+	93
TxtRDD	99+	99+	79	99+
TSP			99+	
WrdCnt	99+	99+	17	93

How Many Runs Needed ...

... to compute average performance with at most 5 % error in 99 % of cases ?

Renaissance 0.10				
bench	C8	C11	E8	E11
aka-uct	3	1	3	9
als	28	2	2	4
chi-sqr	34	22		39
db-shot	6	6	2	2
dec-tre	4	4	7	1
dotty	1	1	1	1
fin-chi	11	22	33	99+
fin-htt	1	1	2	1
fj-kmns	3	3	1	13
fut-gen	5	3	8	5
gauss	21	12	99+	99+
log-reg	3	6	2	3
mne	5	9	14	4
mov-len	1	1	1	1
nai-bay	1	1	95	68
neo-ana	41	5	3	6
pg-rank	7	5	8	1
par-mne	6	5	2	2
philos	10	99+	5	16
reactr	3	2	19	13

rx-scrb	2	3	1	1
sc-doku	99+	72	99+	99+
sc-kmns	1	3	1	1
sc-stmb	4	1	2	6
scrb	20	5	24	76

ScalaBench (with DaCapo)				
bench	C8	C11	E8	E11
	1		1	
appar-d	99+	99+	2	33
avror-l	1	1	1	1
batik-s	1	3	1	1
eclps-s	1		3	
factr-d	8	11	35	70
fop-d	1	9	1	5
h2-d	2	1	1	2
jythn-l	2	12	4	1
kiama-d	1	1	2	2
luidx-d	7	4	1	1
lusrc-l	1	2	4	2
pmd-l	2	2	1	1
scc-l	4	9	1	1
scdoc-l	2	2	2	1

scp-l	7	1	7	2
scrfm-h	2	2	1	1
scxb-h	26	23	28	14
specs-l	5	1	3	1
sunfl-l	1	3	2	4
tmt-d	1	2	1	2
trdb-d	1	3	1	2
trds-l	4	1	1	1
xalan-l	1	1	1	2

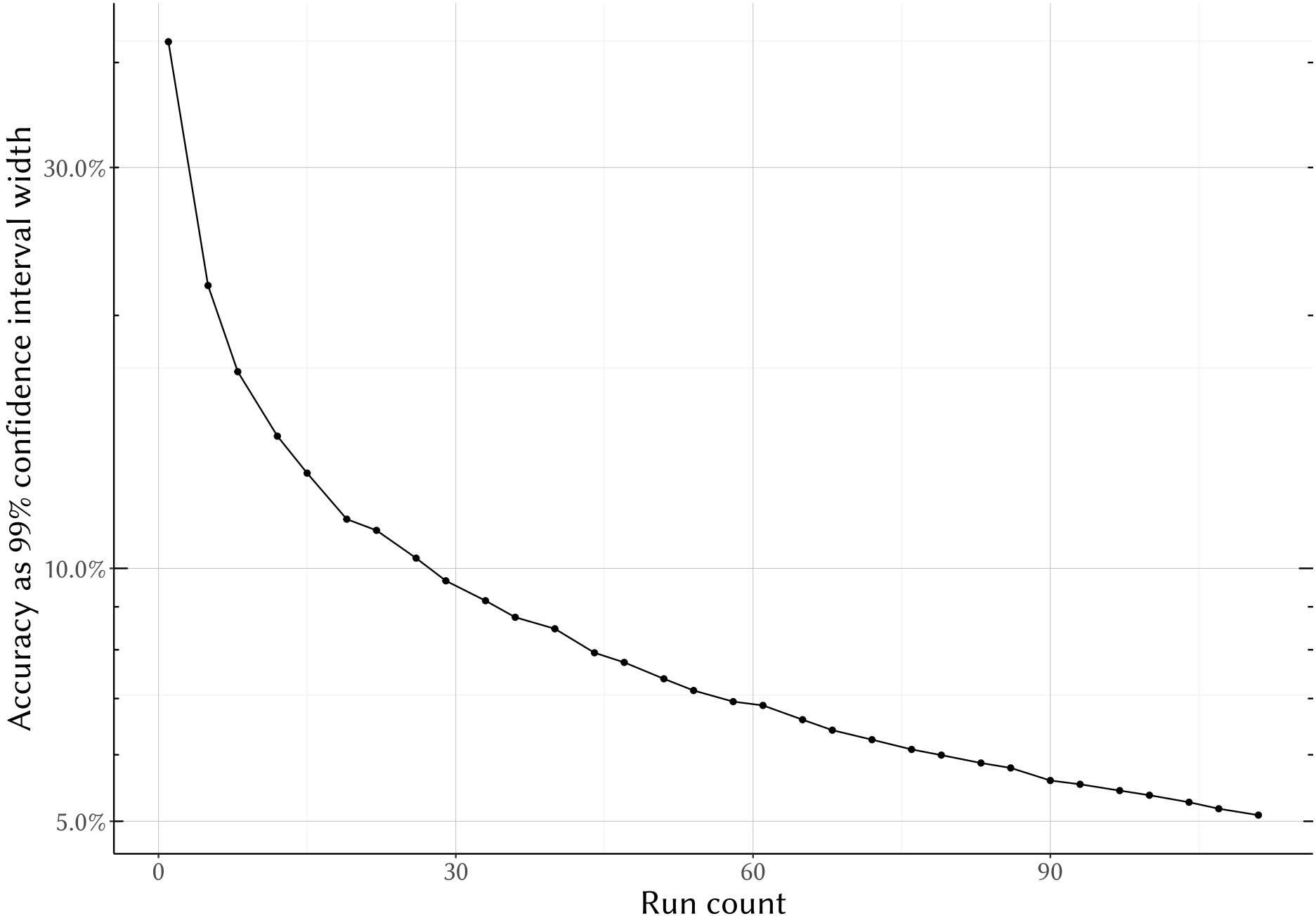
SPECjvm2008 (modified)				
bench	C8	C11	E8	E11
cmp.cmp	1		3	
cmp.sun	2		1	
compr	1	1	1	2
cry.aes	1	1	8	1
cry.rsa	1	1	1	1
cry.sgn	1	1	1	16
derby	1	1	1	5
mpega	1	1	1	1
sci.ffl	31	12	26	6
sci.lul	1	1	1	1
sci.mtc	1	1	9	1

sci.sol	1	1	1	3
sci.spl	1	3	1	1
serial	2	1	2	8
sunflow	1	1	2	1
xml.trn	1	2	1	2
xml.val	8	1	3	7

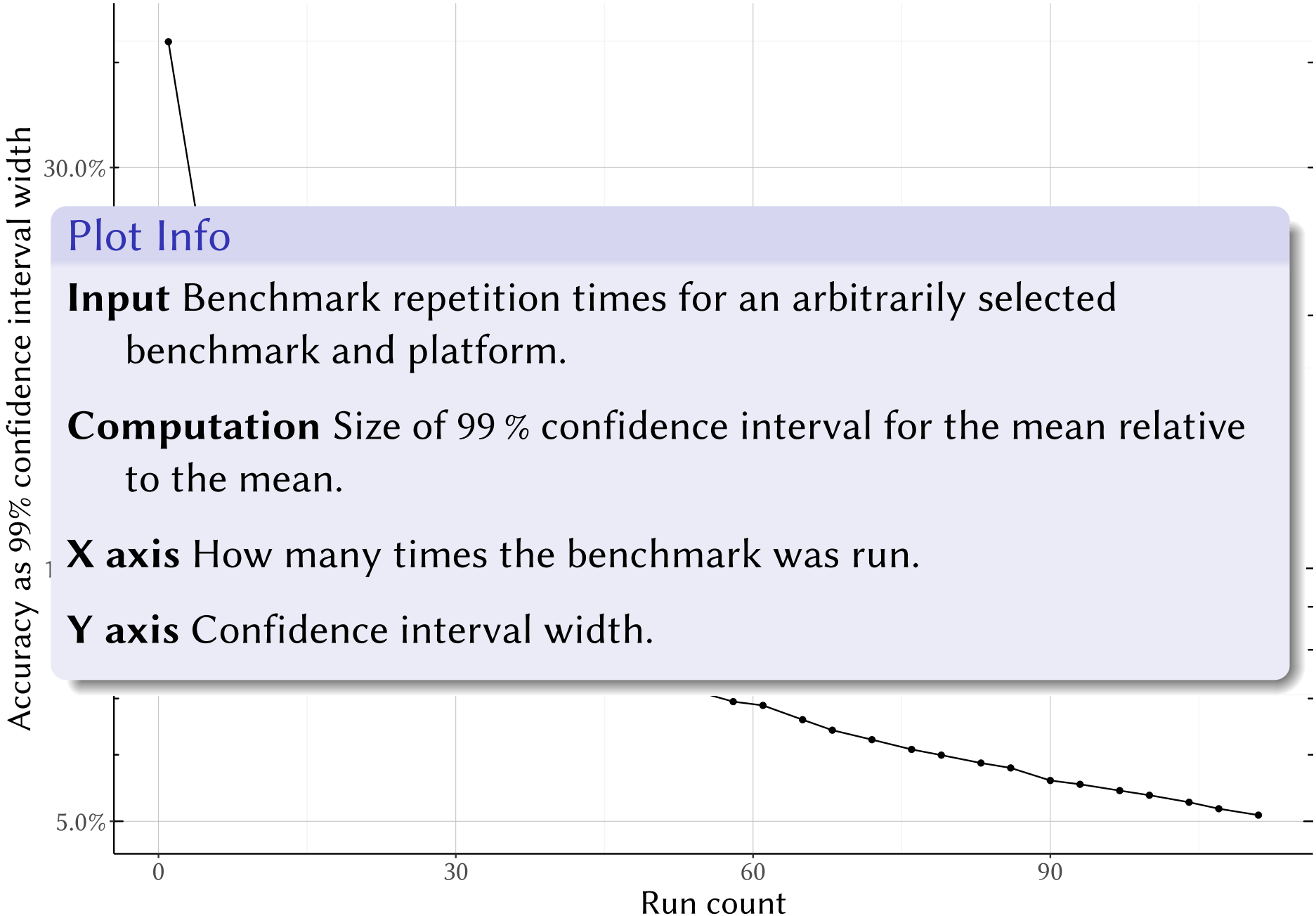
Internal Micros				
bench	C8	C11	E8	E11
BufDec	3	6	1	1
BufEnc	1	27	1	2
ChrHis	13	8	2	3
ChrCnt	14	8	1	20
FltOdd	13	1	1	1
FndNgt	1	6	1	1
FntNgtR	2	1	2	1
FJHis	1	1	1	1
FJStr	2	1	6	3
FldSum	1	4	79	78
FldSumR	1	1	2	1
ForSum	5	5	81	82
ForSumR	10	1	1	9
GrpRem	9	11	9	16

MapOne	99+	17	59	1
NetDot	1	1	1	1
NetEig	1	1	2	1
Reduce	6	14	13	43
STMLst	17	8	7	5
STMMap	19	15	6	4
Scan	5	14	20	99+
SrtRDD	3	9	6	20
StdDev	99+	99+	99+	1
StrCnt	5	5	1	37
StrDem	99+	26	10	16
StrDev	1	1	1	7
SFndNeg	1	2	62	6
SFldSum	42	1	11	8
SForSum	1	1	1	30
SMapRed	59	50	2	6
StrPer	9	2	99+	3
STwoAvg	2	35	10	17
TxtSDF	12	2	4	8
TxtRDD	11	12	2	55
TSP			42	
WrdCnt	5	9	11	3

How Accuracy Relates To Run Count ?



How Accuracy Relates To Run Count ?



Plot Info

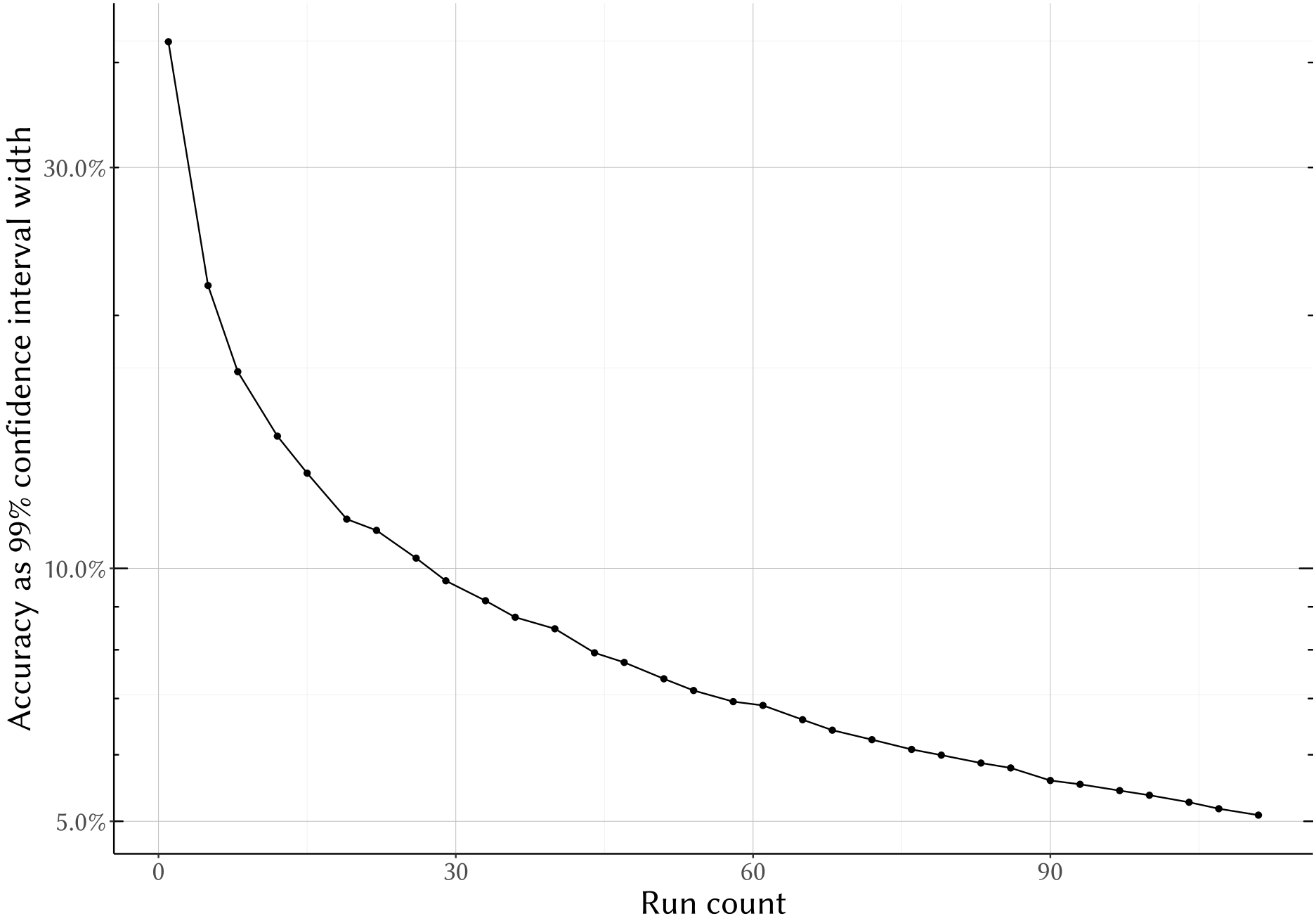
Input Benchmark repetition times for an arbitrarily selected benchmark and platform.

Computation Size of 99 % confidence interval for the mean relative to the mean.

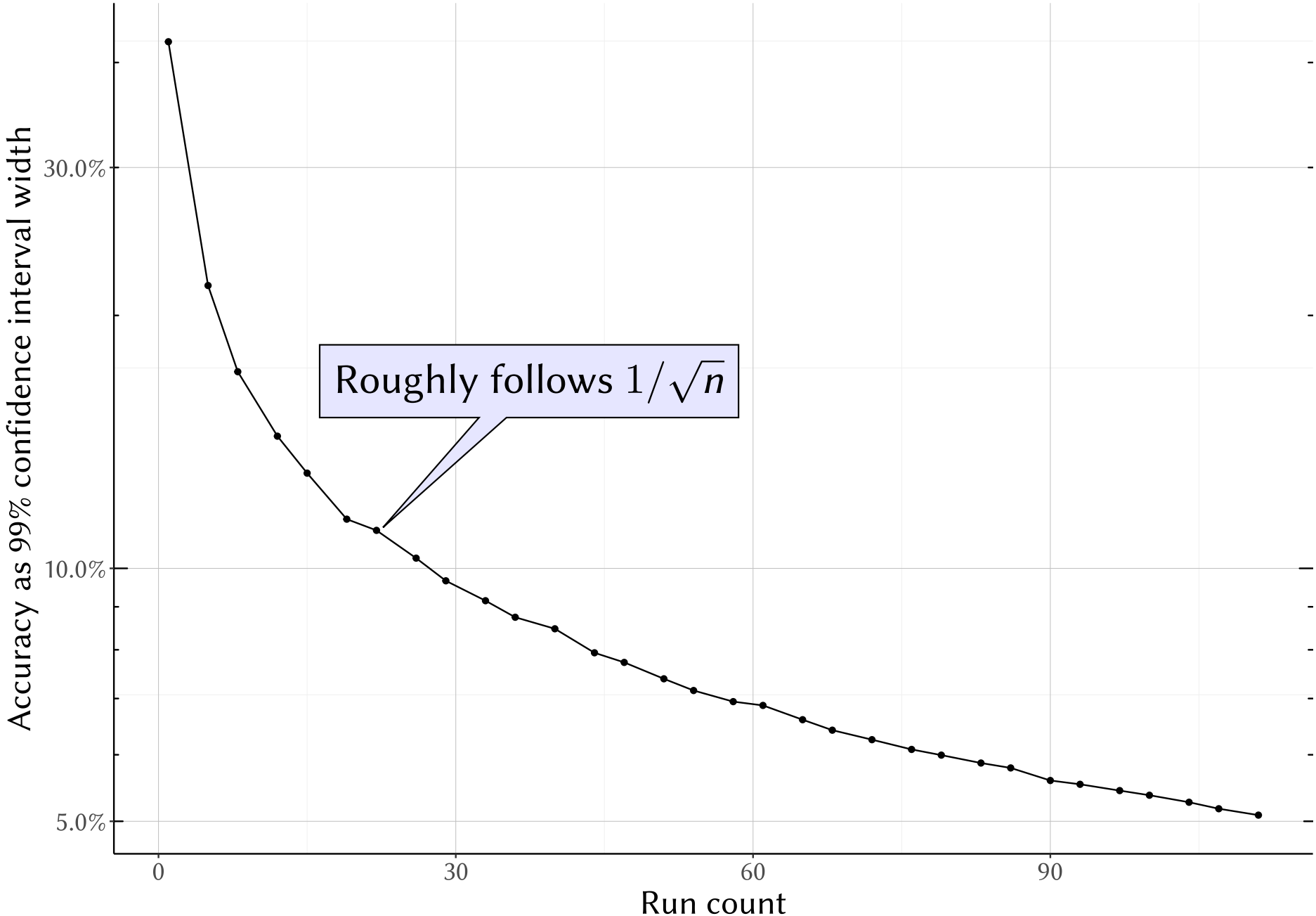
X axis How many times the benchmark was run.

Y axis Confidence interval width.

How Accuracy Relates To Run Count ?



How Accuracy Relates To Run Count ?



Take Away So Far ...

Running benchmarks only once may not be enough

- Non deterministic compilation especially with microbenchmarks
- But the presented tables also include simple cases of high variance

Starting with accuracy as sole goal is not good

- Aiming for excessive accuracy backfires quickly
- Conditions impacting accuracy may change
- How to define balance ?

Accuracy is a function of more than just the benchmark

Outline

Runs Needed When Different Metrics Used ...

... to compute average performance with at most 1 % error in 99 % of cases.

Renaissance 0.10			
bench	time	clk	ins
aka-uct	99+	99+	99+
als	99+	99+	99+
chi-sqr	99+	99+	99+
db-shot	99+	99+	99+
dec-tre	73	84	78
dotty	17	18	16
fin-chi	99+	99+	99+
fin-htt	58	99+	33
fj-kmns	12	14	13
fut-gen	99+	99+	99+
gauss	99+	99+	99+
log-reg	34	35	99+
mne	99+	99+	64
mov-len	6	13	14
nai-bay	5	6	16
neo-ana	99+	99+	99+
pg-rank	99+	99+	99+
par-mne	78	80	57
philos	99+	99+	98
reactr	99+	99+	99+

rx-scrb	75	67	54
sc-doku	99+	99+	99+
sc-kmns	53	53	99+
sc-stmb	71	99+	99+
scrb	99+	99+	99+
ScalaBench (with DaCapo)			
bench	time	clk	ins
	59	99+	99+
appar-d	99+	99+	99+
avror-l	5	24	91
batik-s	1	1	1
eclps-s	3	16	3
factr-d	99+	99+	99+
fop-d	18	17	15
h2-d	27	15	12
jythn-l	31	32	12
kiama-d	37	60	56
luidx-d	94	28	1
lusrc-l	38	49	36
pmd-l	44	16	11
scc-l	65	72	99+
scdoc-l	54	65	99+

scp-l	23	94	69
scrfm-h	33	69	75
scxb-h	99+	99+	99+
specs-l	16	26	9
sunfl-l	6	6	8
tmt-d	13	18	78
trdb-d	14	99+	99+
trds-l	7	12	7
xalan-l	32	99+	99+
SPECjvm2008 (modified)			
bench	time	clk	ins
cmp.cmp	20	21	36
cmp.sun	8	7	11
compr	9	9	4
cry.aes	11	11	7
cry.rsa	6	14	1
cry.sgn	12	12	25
derby	23	23	5
mpega	1	1	1
sci.ffl	99+	99+	99+
sci.lul	1	1	1
sci.mtc	12	12	22

sci.sol	1	1	1
sci.spl	1	1	1
serial	12	12	3
sunflow	7	6	11
xml.trn	9	9	2
xml.val	20	26	1
Internal Micros			
bench	time	clk	ins
BufDec	4	3	1
BufEnc	5	8	1
ChrHis	99+	99+	53
ChrCnt	99+	99+	89
FltOdd	99+	99+	99+
FndNgt	1	1	15
FntNgtR	2	2	1
FJHis	6	9	57
FJStr	15	12	8
FldSum	4	3	1
FldSumR	1	1	1
ForSum	99+	99+	1
ForSumR	99+	99+	1
GrpRem	57	58	99+

MapOne	99+	99+	99+
NetDot	1	1	1
NetEig	1	1	1
Reduce	99+	99+	72
STMLst	99+	99+	99+
STMMap	99+	99+	99+
Scan	73	99+	44
SrtRDD	67	99+	26
StdDev	99+	99+	99+
StrCnt	79	99+	64
StrDem	99+	99+	99+
StrDev	1	1	1
SFndNeg	1	1	99+
SFldSum	44	44	42
SForSum	1	1	1
SMapRed	99+	99+	99+
StrPer	99+	99+	31
STwoAvg	27	27	12
TxtSDF	99+	99+	25
TxtRDD	99+	99+	31
WrdCnt	99+	99+	8

time - wall clock time

clk - thread clock time

ins - instruction count

Runs Needed When Different Metrics Used ...

... to compute average performance with at most 1 % error in 99 % of cases.

Renaissance 0.10			
bench	time	clk	ins
aka-uct	99+	99+	99+
als	99+	99+	99+
chi-sqr	99+	99+	99+
db-shot	99+	99+	99+
dec-tre	73	84	78
dotty	17	18	16
fin-chi	99+	99+	99+
fin-htt	58	99+	33
fj-kmns	12	14	13
fut-gen	99+	99+	99+
gauss	99+	99+	99+
log-reg	34	35	99+
mne	99+	99+	64
mov-len	6	13	14
nai-bay	5	6	16
neo-ana	99+	99+	99+
pg-rank	99+	99+	99+
par-mne	78	80	57
philos	99+	99+	98
reactr	99+	99+	99+

rx-scrb	75	67	54
sc-doku	99+	99+	99+
sc-kmns	53	53	99+
sc-stmb	71	99+	99+
scrb	99+	99+	99+
ScalaBench (with DaCapo)			
bench	time	clk	ins
	59	99+	99+
appar-d	99+	99+	99+
avror-l	5	24	91
batik-s	1	1	1
eclps-s	3	16	3
factr-d	99+	99+	99+
fop-d	18	17	15
h2-d	27	15	12
jythn-l	31	32	12
kiama-d	37	60	56
luidx-d	94	28	1
lusrc-l	38	49	36
pmd-l			
scc-l			
scdoc-l			

scp-l	23	94	69
scrfm-h	33	69	75
scxb-h	99+	99+	99+
specs-l	16	26	9
sunfl-l	6	6	8
tmt-d	13	18	78
trdb-d	14	99+	99+
trds-l	7	12	7
xalan-l	32	99+	99+
SPECjvm2008 (modified)			
bench	time	clk	ins
cmp.cmp	20	21	36
cmp.sun	8	7	11
compr	9	9	4
cry.aes	11	11	7
cry.rsa	6	14	1
cry.sgn	12	12	25
derby	23	23	5
ega	1	1	1

sci.sol	1	1	1
sci.spl	1	1	1
serial	12	12	3
sunflow	7	6	11
xml.trn	9	9	2
xml.val	20	26	1
Internal Micros			
bench	time	clk	ins
BufDec	4	3	1
BufEnc	5	8	1
ChrHis	99+	99+	53
ChrCnt	99+	99+	89
FltOdd	99+	99+	99+
FndNgt	1	1	15
FntNgtR	2	2	1
FJHis	6	9	57
FJStr	15	12	8
FldSum	4	3	1
FldSumR	1	1	1
			1
			1
		58	99+

MapOne	99+	99+	99+
NetDot	1	1	1
NetEig	1	1	1
Reduce	99+	99+	72
STMLst	99+	99+	99+
STMMap	99+	99+	99+
Scan	73	99+	44
SrtRDD	67	99+	26
StdDev	99+	99+	99+
StrCnt	79	99+	64
StrDem	99+	99+	99+
StrDev	1	1	1
SFndNeg	1	1	99+
SFldSum	44	44	42
SForSum	1	1	1
SMapRed	99+	99+	99+
StrPer	99+	99+	31
STwoAvg	27	27	12
TxtSDF	99+	99+	25
TxtRDD	99+	99+	31
WrdCnt	99+	99+	8

Instruction count quite stable even when time is not

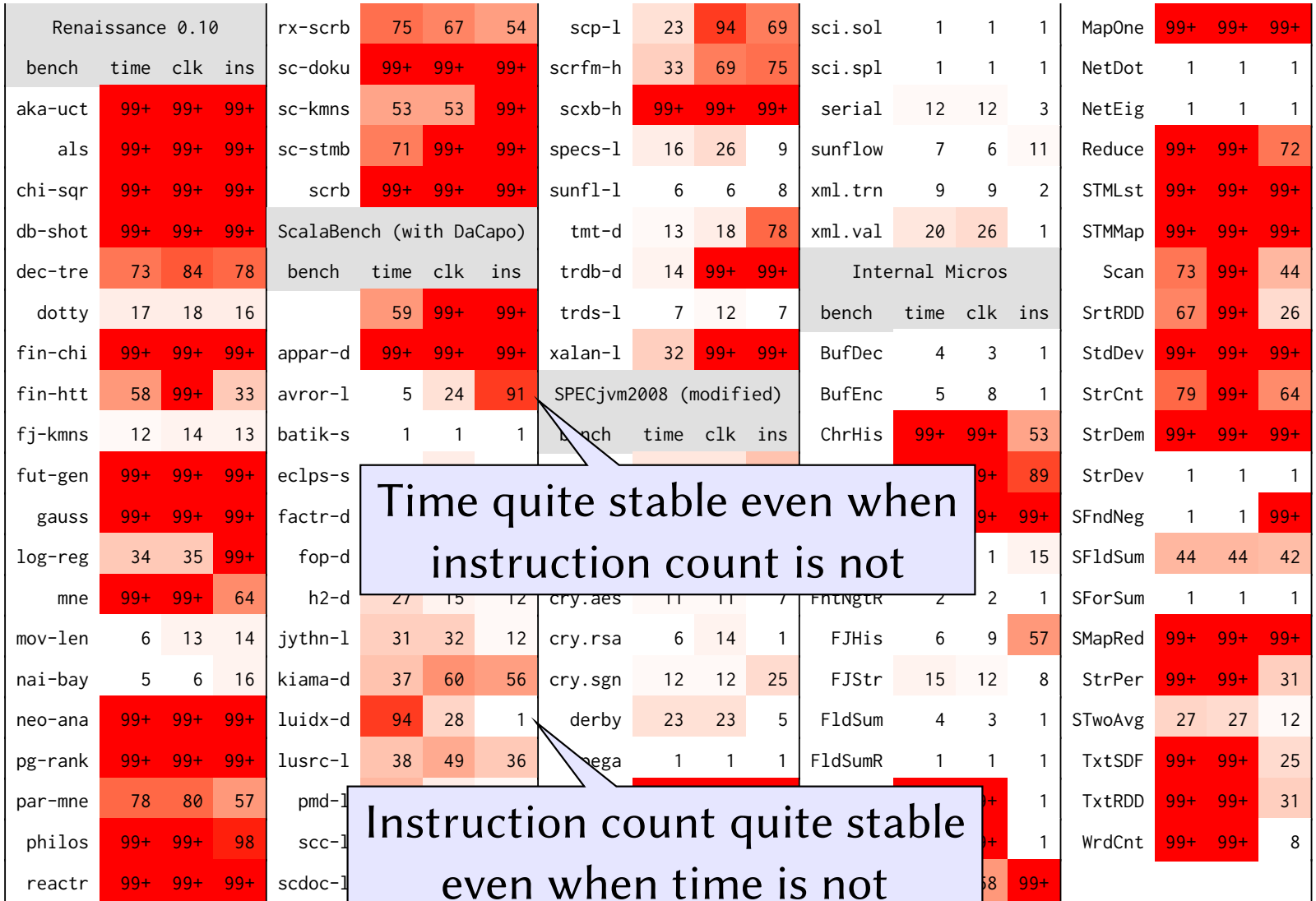
time - wall clock time

clk - thread clock time

ins - instruction count

Runs Needed When Different Metrics Used ...

... to compute average performance with at most 1 % error in 99 % of cases.



Renaissance 0.10			
bench	time	clk	ins
aka-uct	99+	99+	99+
als	99+	99+	99+
chi-sqr	99+	99+	99+
db-shot	99+	99+	99+
dec-tre	73	84	78
dotty	17	18	16
fin-chi	99+	99+	99+
fin-htt	58	99+	33
fj-kmns	12	14	13
fut-gen	99+	99+	99+
gauss	99+	99+	99+
log-reg	34	35	99+
mne	99+	99+	64
mov-len	6	13	14
nai-bay	5	6	16
neo-ana	99+	99+	99+
pg-rank	99+	99+	99+
par-mne	78	80	57
philos	99+	99+	98
reactr	99+	99+	99+

rx-scrb	75	67	54
sc-doku	99+	99+	99+
sc-kmns	53	53	99+
sc-stmb	71	99+	99+
scrb	99+	99+	99+

ScalaBench (with DaCapo)			
bench	time	clk	ins
	59	99+	99+
appar-d	99+	99+	99+
avror-l	5	24	91
batik-s	1	1	1
eclps-s			
factr-d			
fop-d			
h2-d	27	15	12
jythn-l	31	32	12
kiama-d	37	60	56
luidx-d	94	28	1
lusrc-l	38	49	36
pmd-l			
scc-l			
scdoc-l			

scp-l	23	94	69
scrfm-h	33	69	75
scxb-h	99+	99+	99+
specs-l	16	26	9
sunfl-l	6	6	8
tmt-d	13	18	78
trdb-d	14	99+	99+
trds-l	7	12	7
xalan-l	32	99+	99+

SPECjvm2008 (modified)			
bench	time	clk	ins
cry.aes	11	11	7
cry.rsa	6	14	1
cry.sgn	12	12	25
derby	23	23	5
mpega	1	1	1

sci.sol	1	1	1
sci.spl	1	1	1
serial	12	12	3
sunflow	7	6	11
xml.trn	9	9	2
xml.val	20	26	1

Internal Micros			
bench	time	clk	ins
BufDec	4	3	1
BufEnc	5	8	1
ChrHis	99+	99+	53
		9+	89
		9+	99+
		1	15
FntNgtR	2	2	1
FJHis	6	9	57
FJStr	15	12	8
FldSum	4	3	1
FldSumR	1	1	1
		9+	1
		9+	1
	58	99+	

MapOne	99+	99+	99+
NetDot	1	1	1
NetEig	1	1	1
Reduce	99+	99+	72
STMLst	99+	99+	99+
STMMap	99+	99+	99+
Scan	73	99+	44
SrtRDD	67	99+	26
StdDev	99+	99+	99+
StrCnt	79	99+	64
StrDem	99+	99+	99+
StrDev	1	1	1
SFndNeg	1	1	99+
SFldSum	44	44	42
SForSum	1	1	1
SMapRed	99+	99+	99+
StrPer	99+	99+	31
STwoAvg	27	27	12
TxtSDF	99+	99+	25
TxtRDD	99+	99+	31
WrdCnt	99+	99+	8

Time quite stable even when instruction count is not

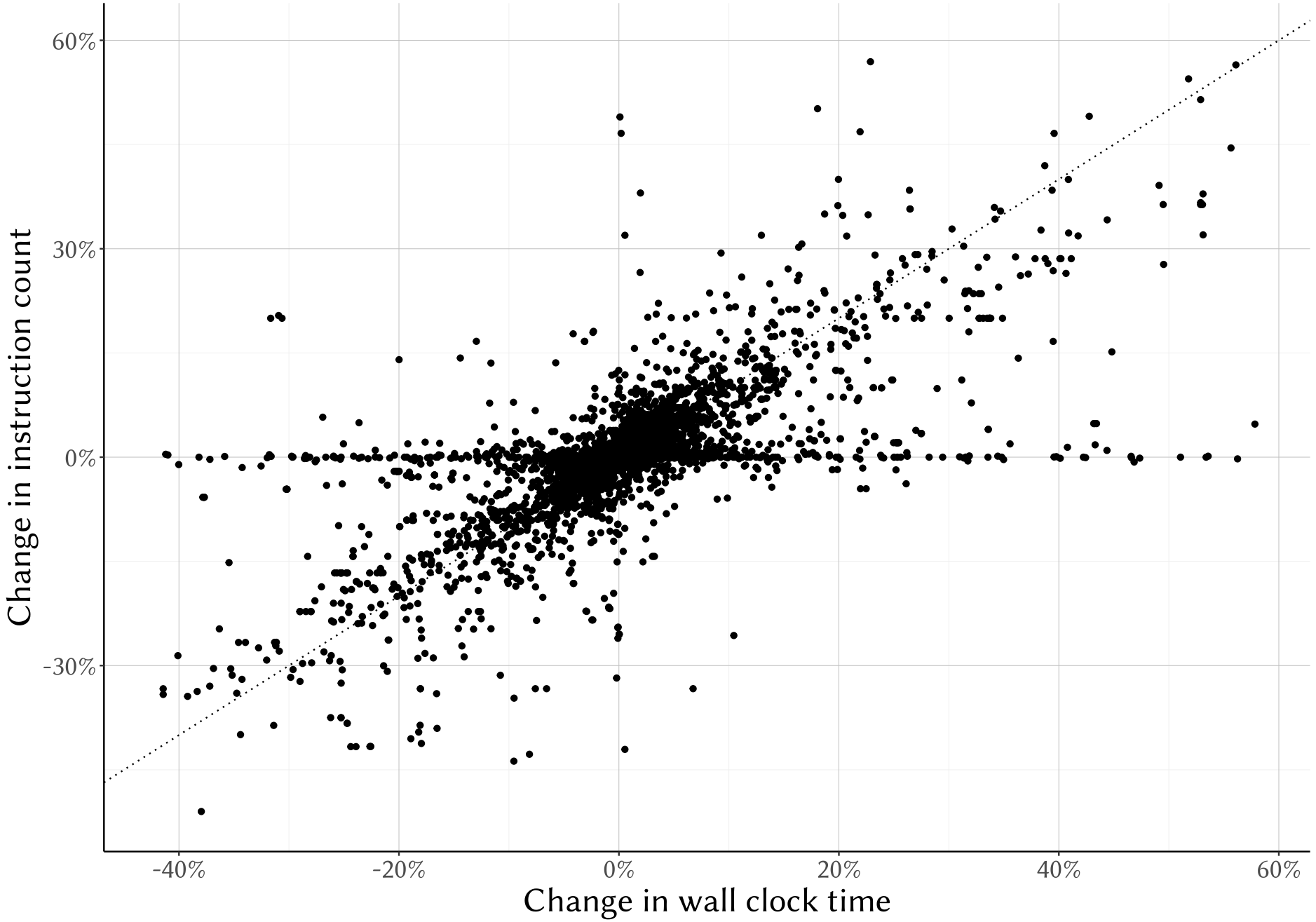
Instruction count quite stable even when time is not

time - wall clock time

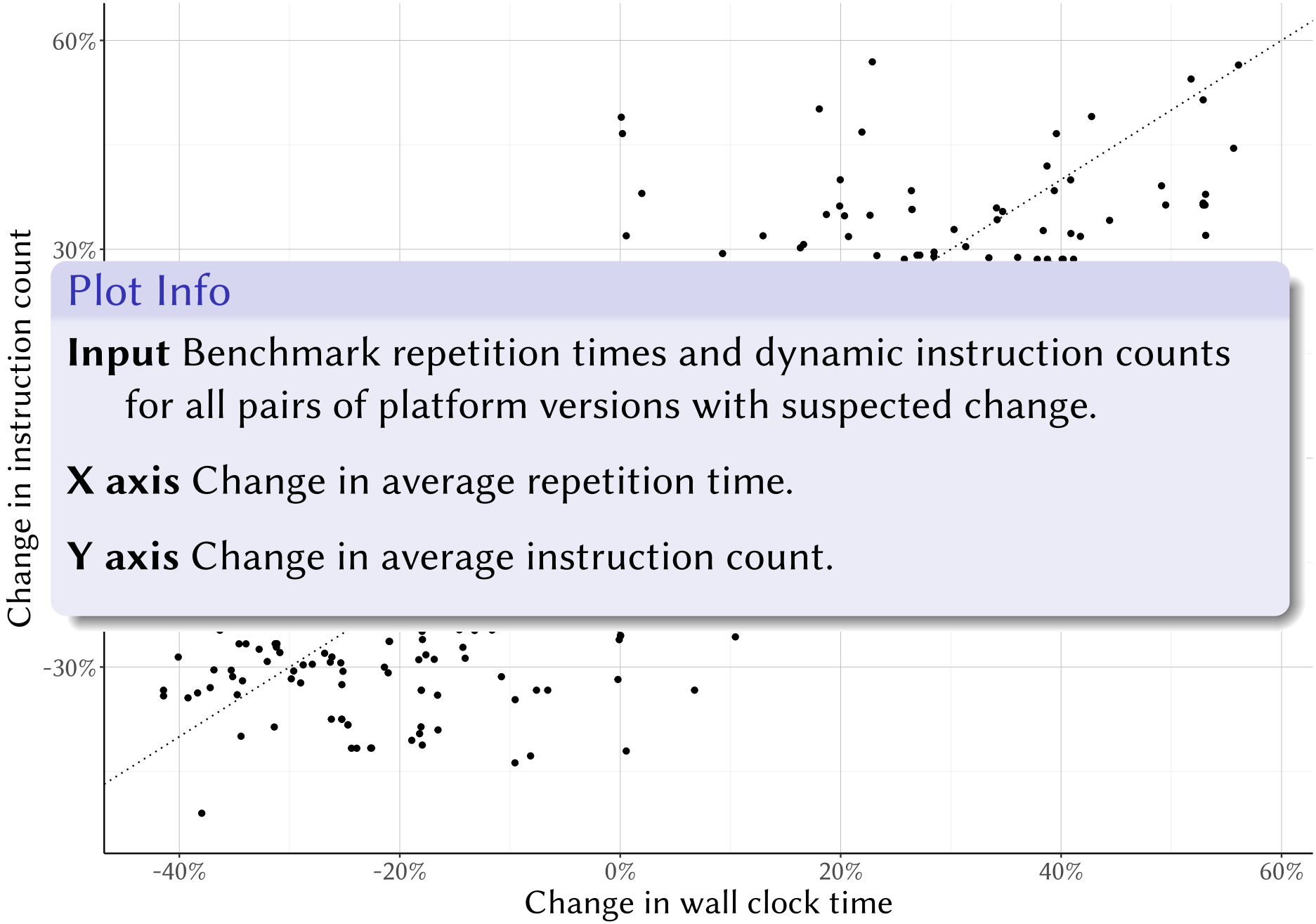
clk - thread clock time

ins - instruction count

Different Metrics Not Always In Sync



Different Metrics Not Always In Sync



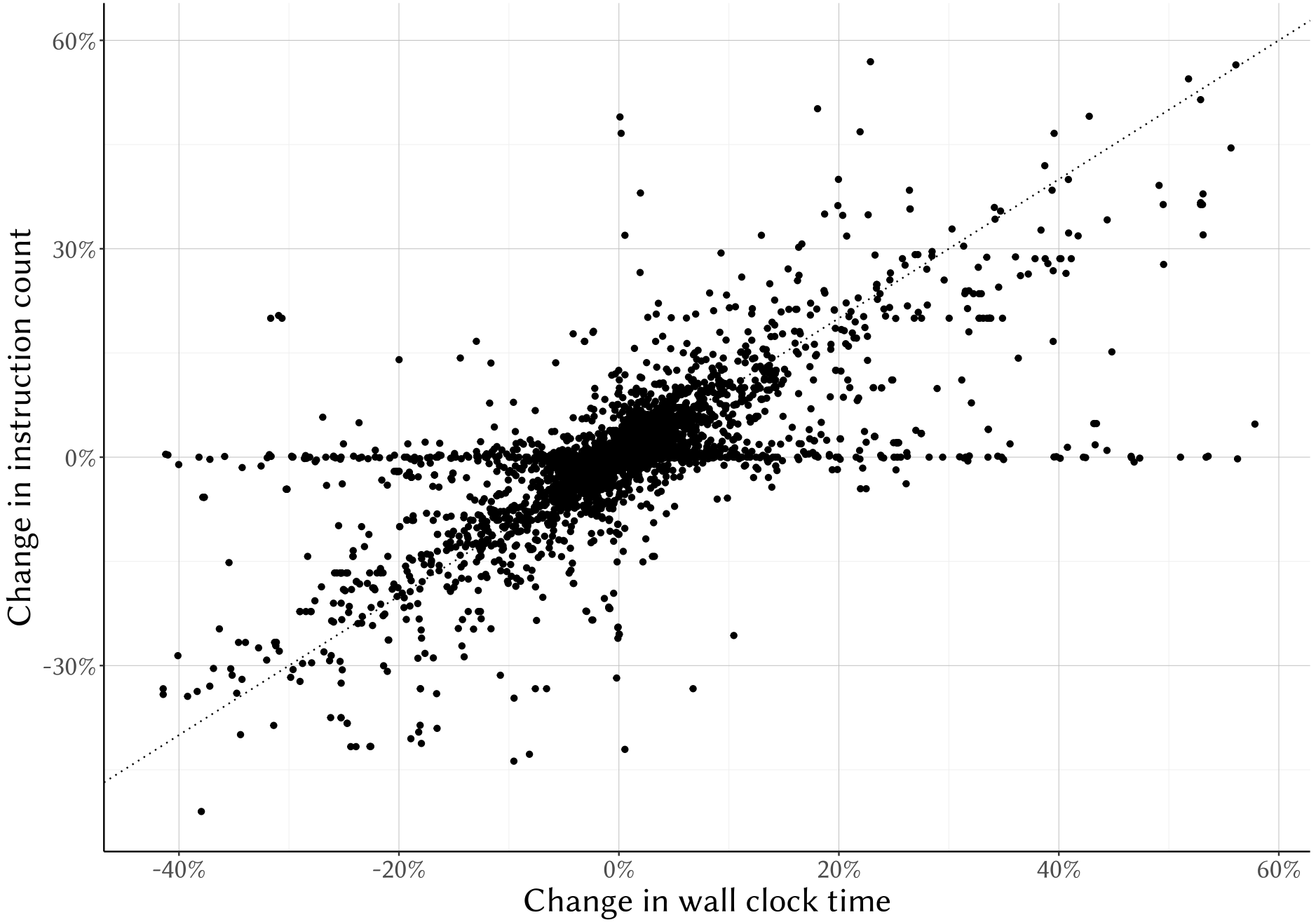
Plot Info

Input Benchmark repetition times and dynamic instruction counts for all pairs of platform versions with suspected change.

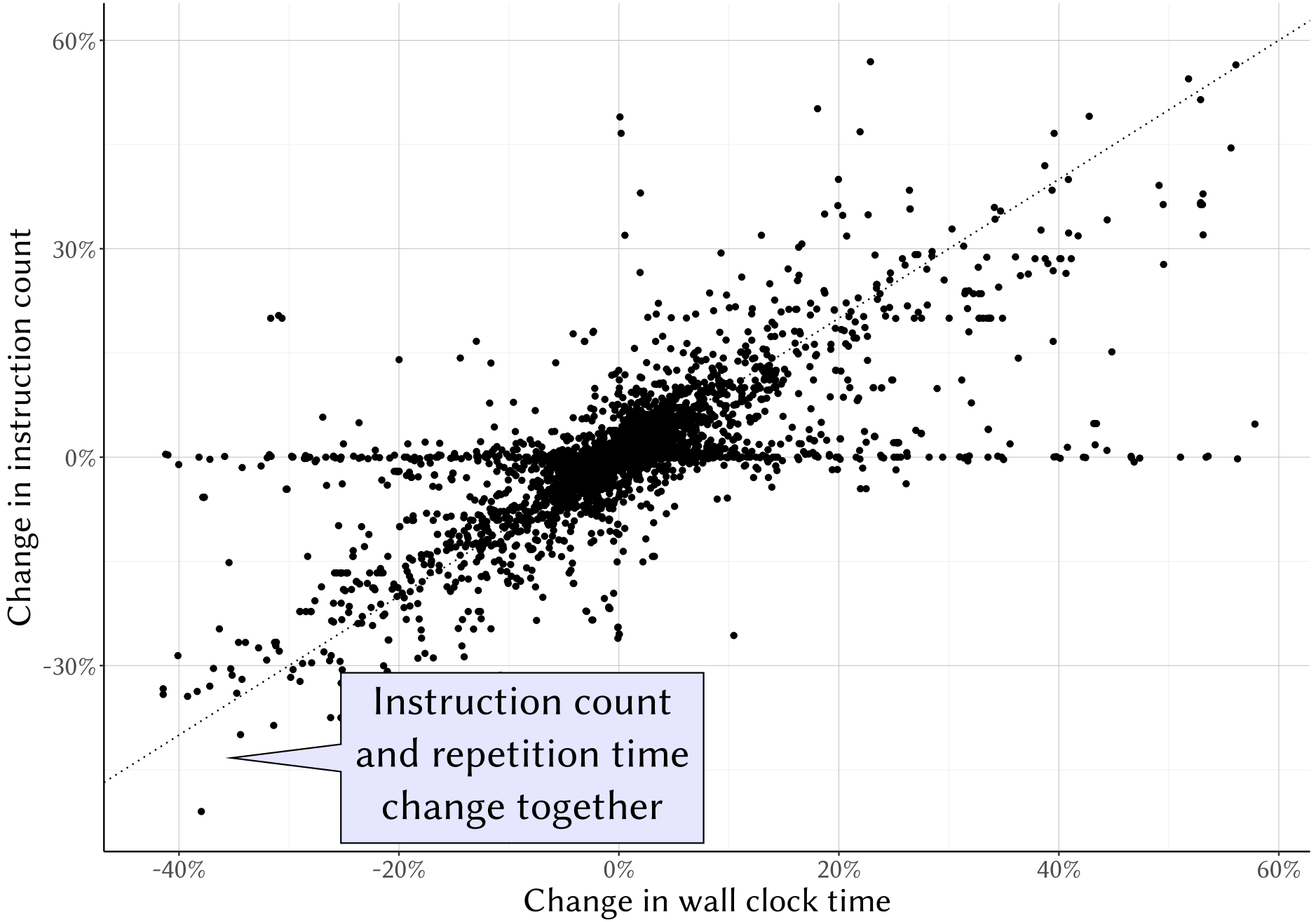
X axis Change in average repetition time.

Y axis Change in average instruction count.

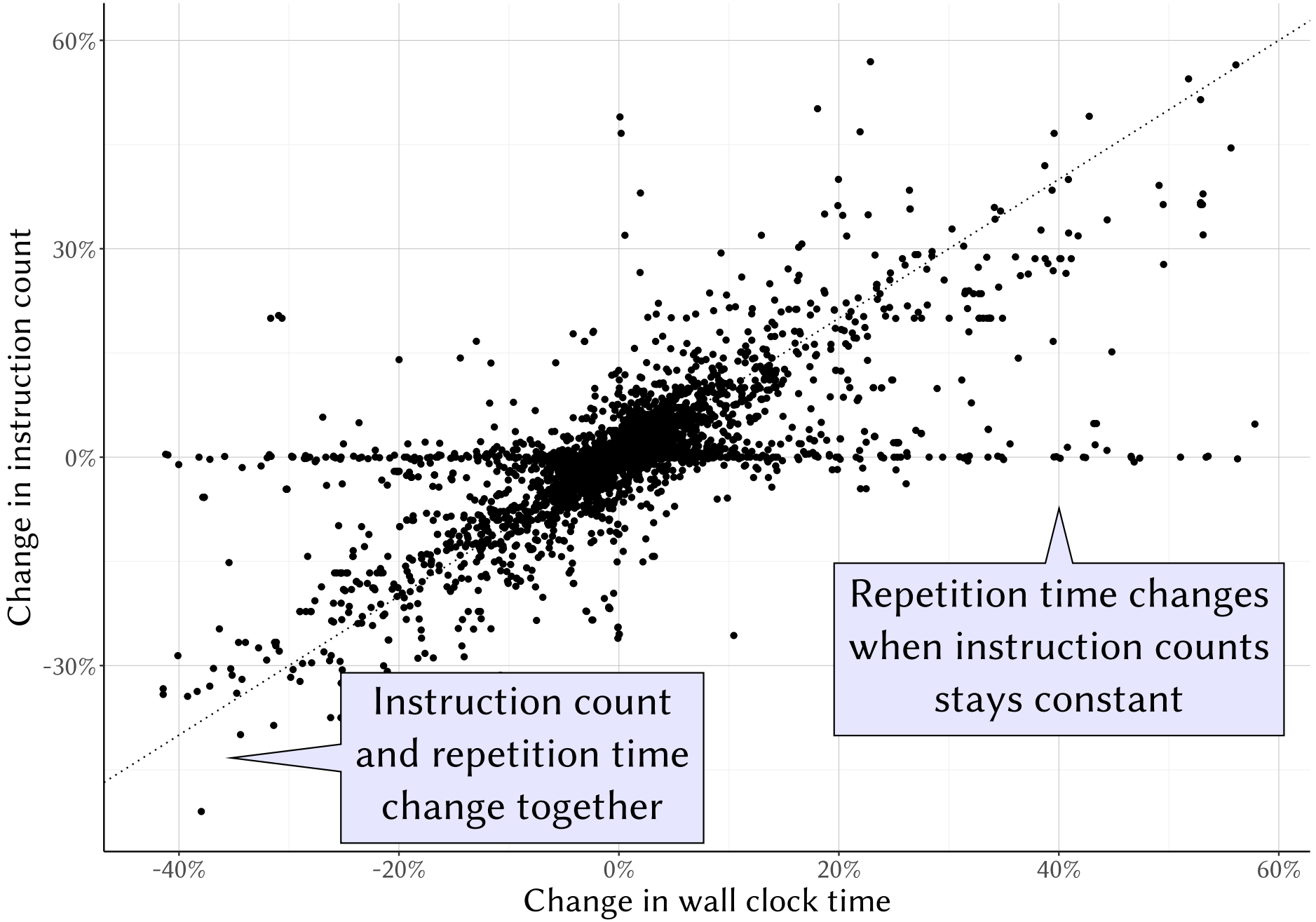
Different Metrics Not Always In Sync



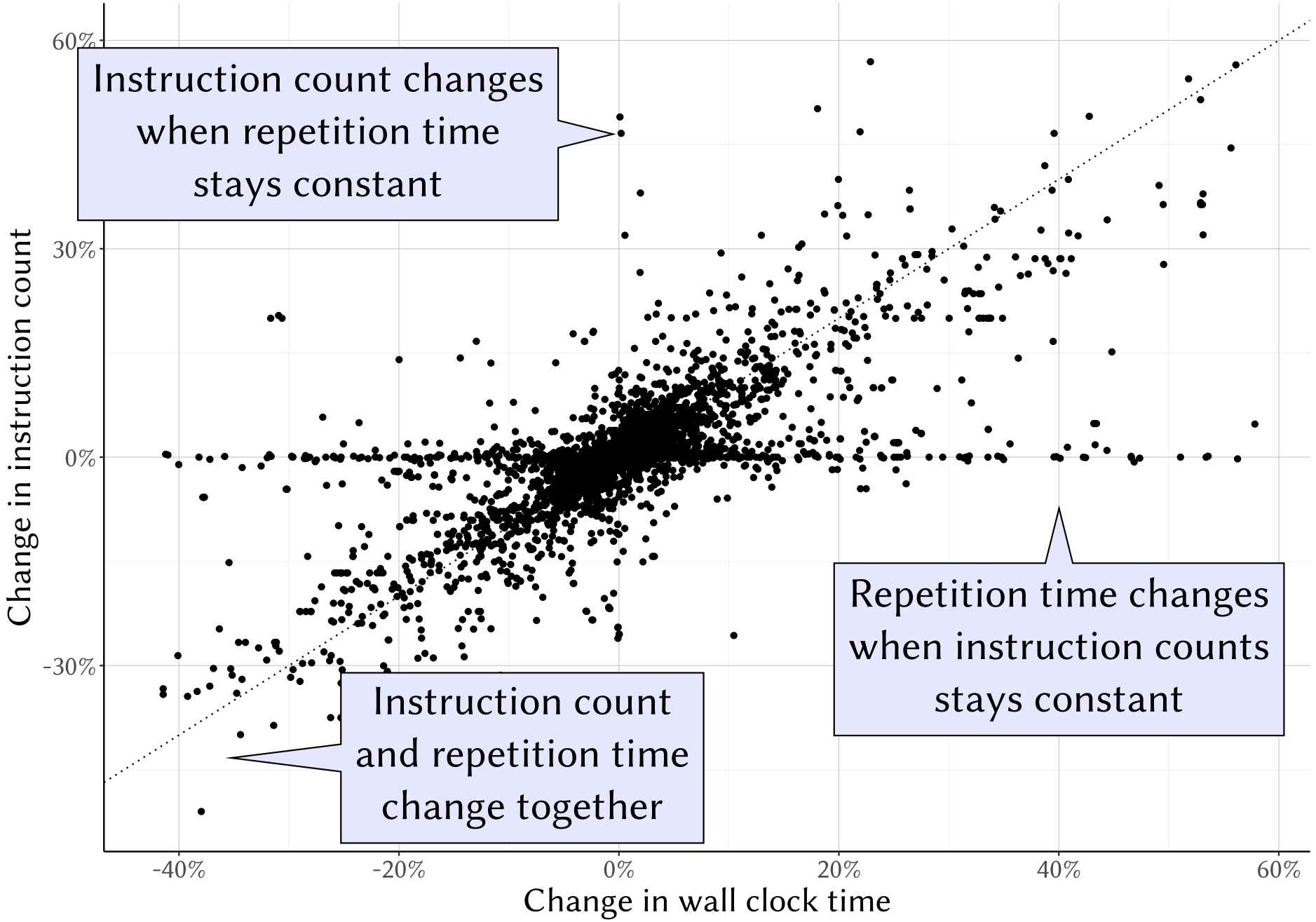
Different Metrics Not Always In Sync



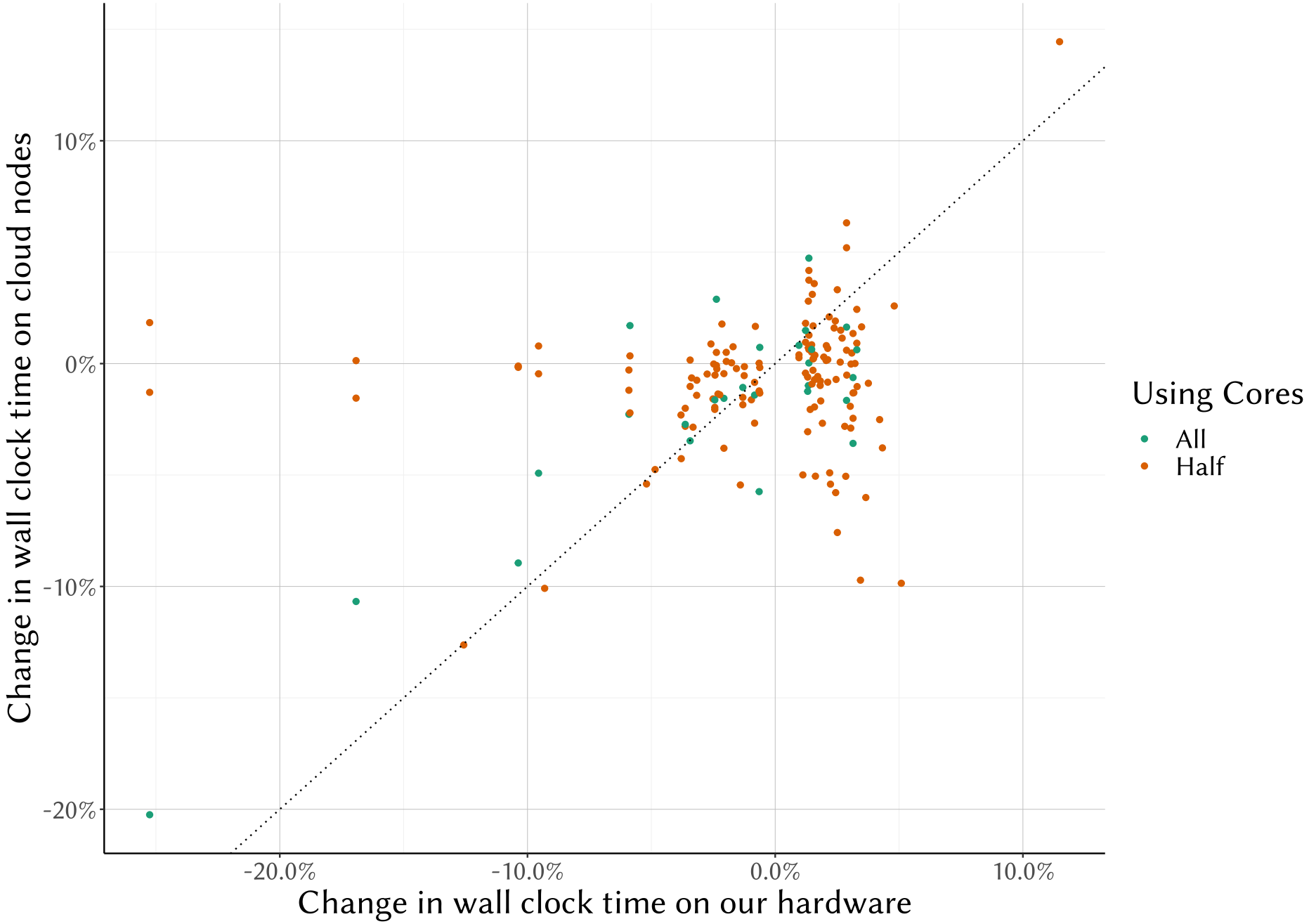
Different Metrics Not Always In Sync



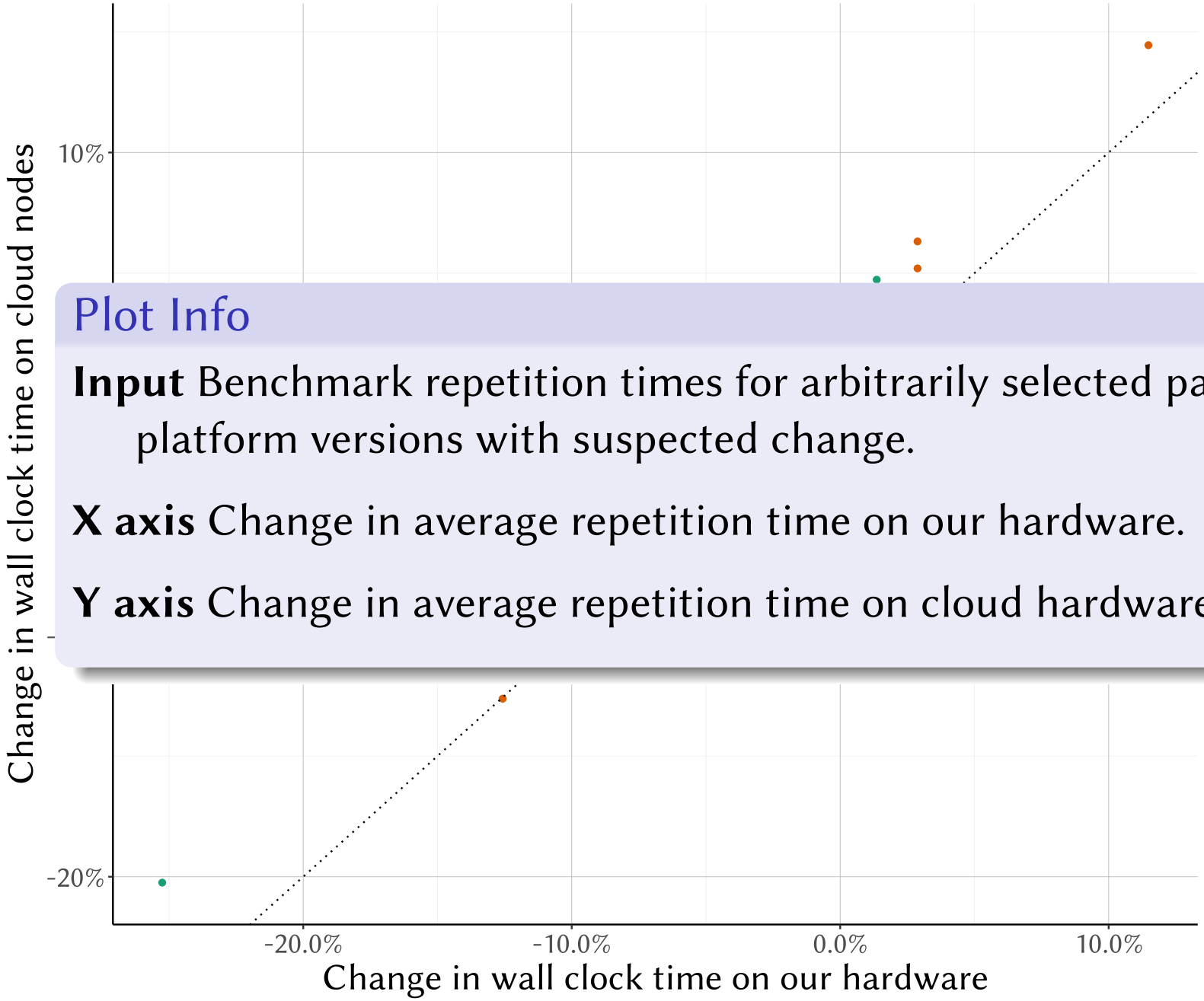
Different Metrics Not Always In Sync



Wall Clock Time Changes Not Always Portable



Wall Clock Time Changes Not Always Portable



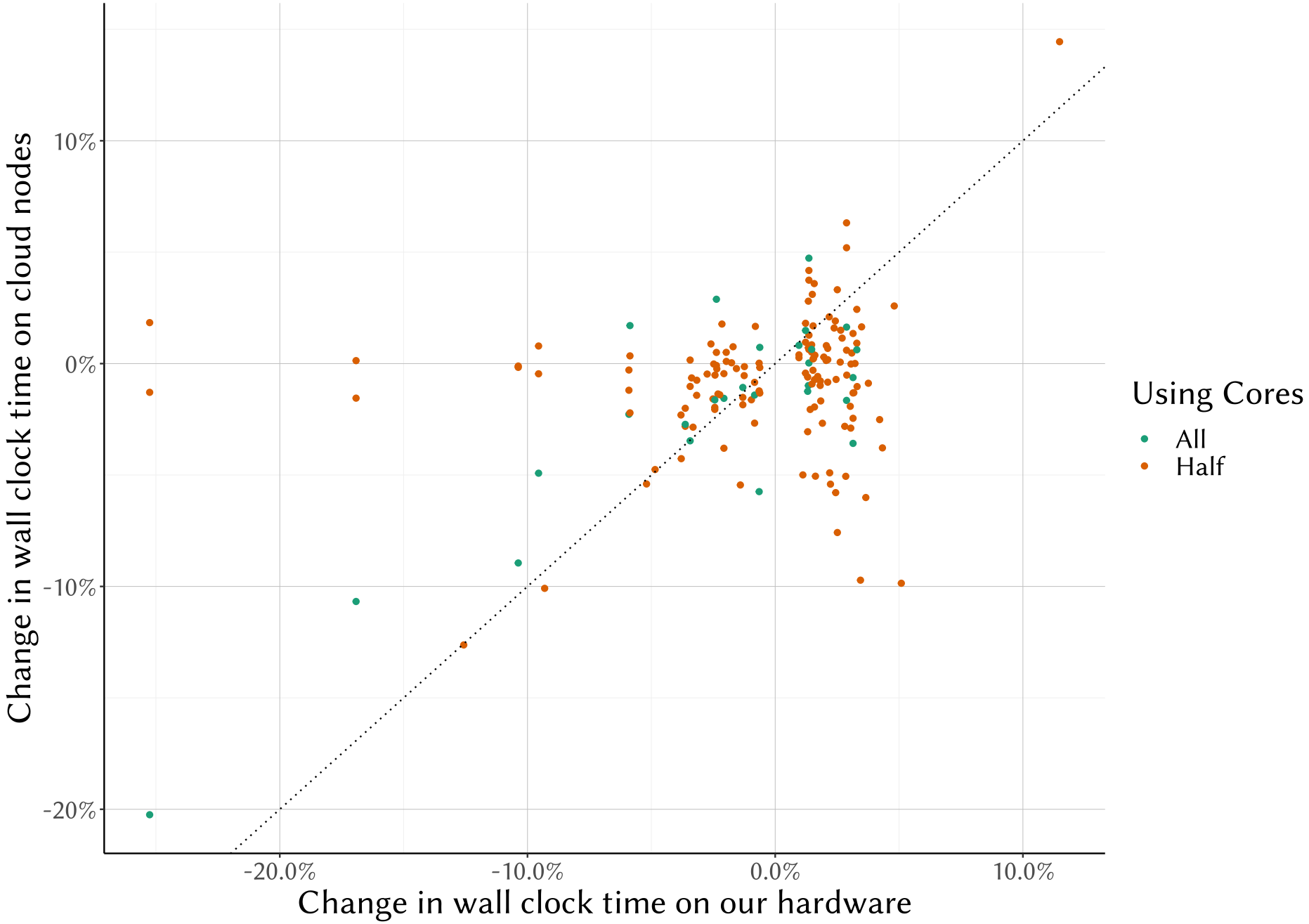
Plot Info

Input Benchmark repetition times for arbitrarily selected pairs of platform versions with suspected change.

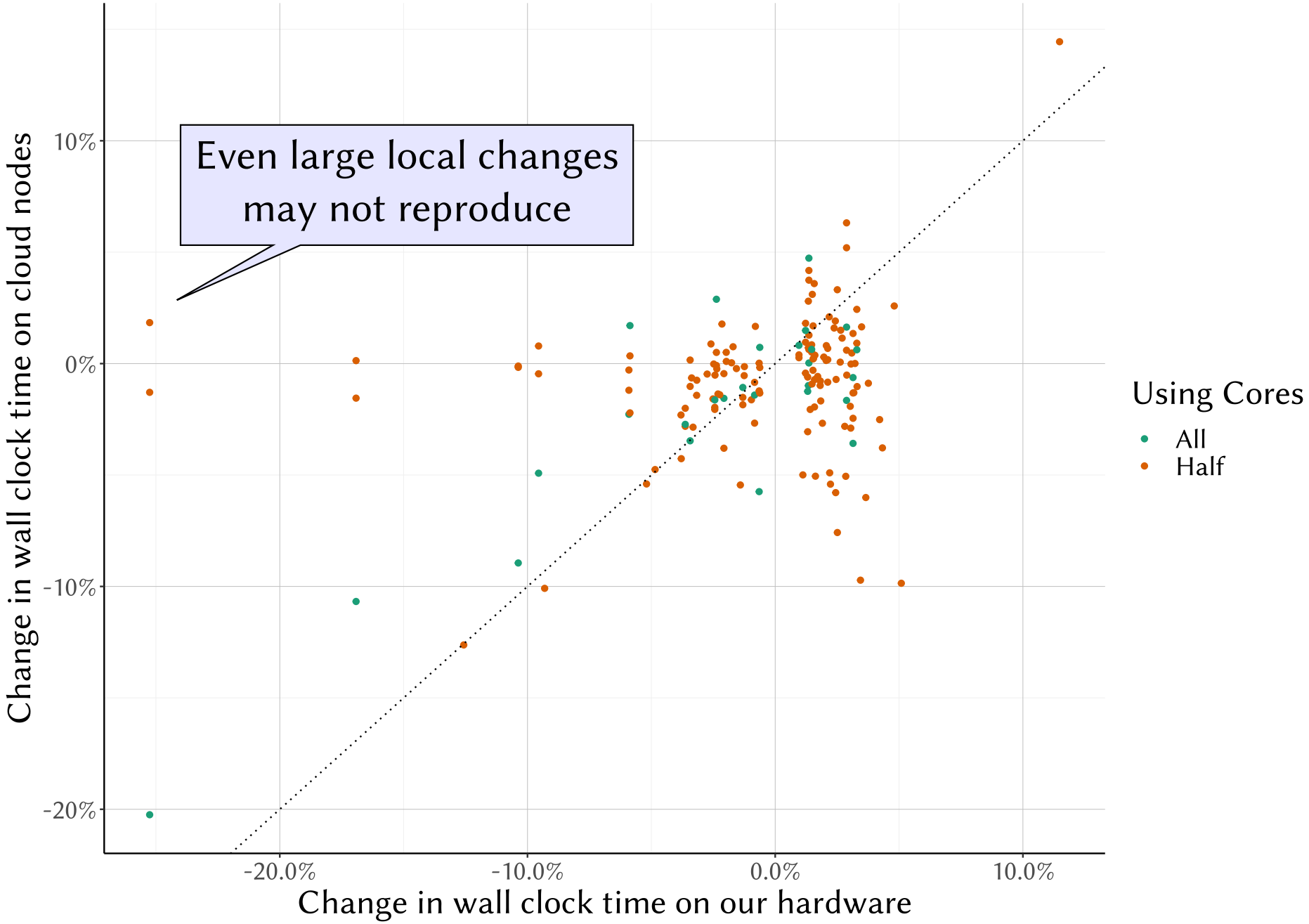
X axis Change in average repetition time on our hardware.

Y axis Change in average repetition time on cloud hardware.

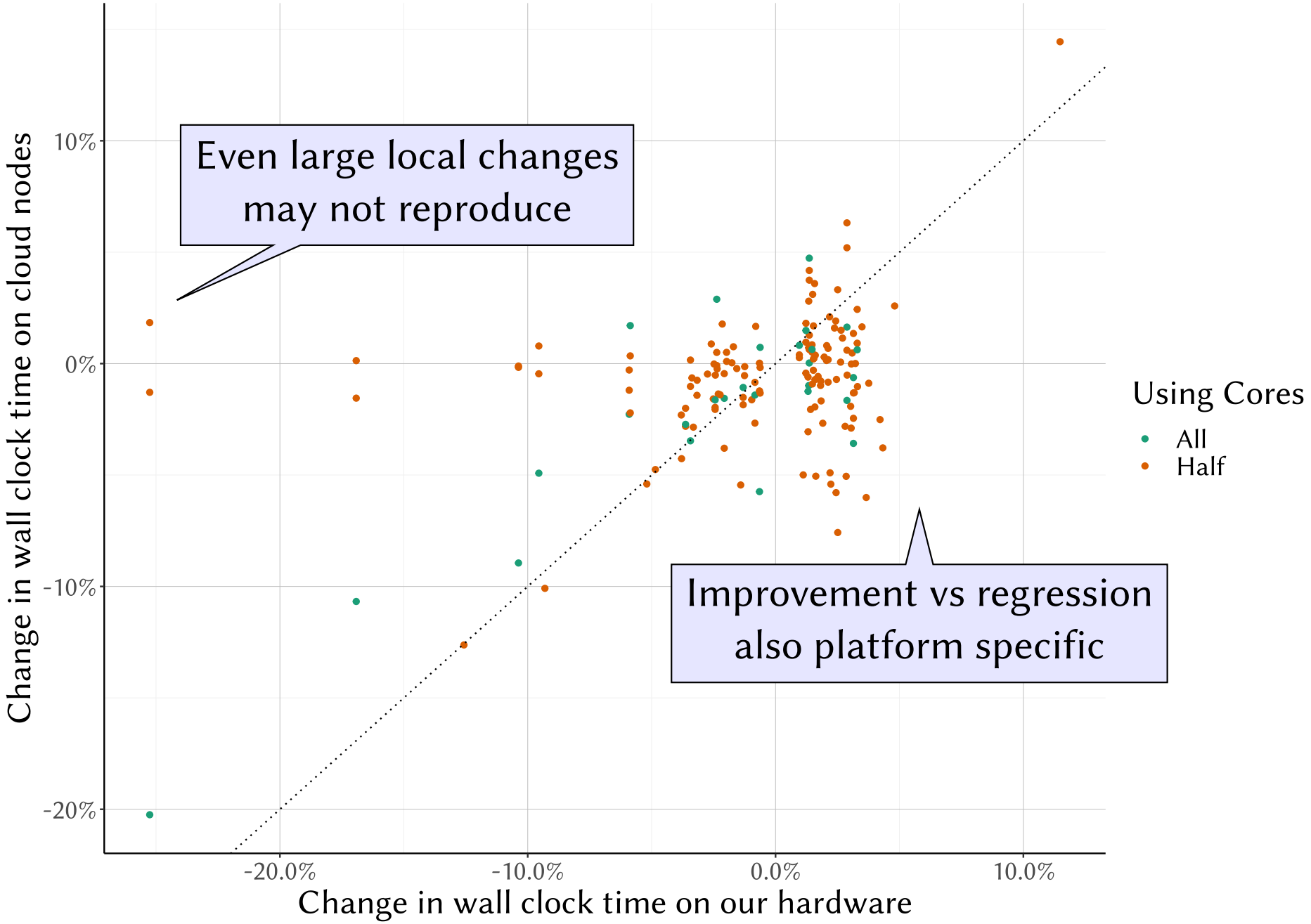
Wall Clock Time Changes Not Always Portable



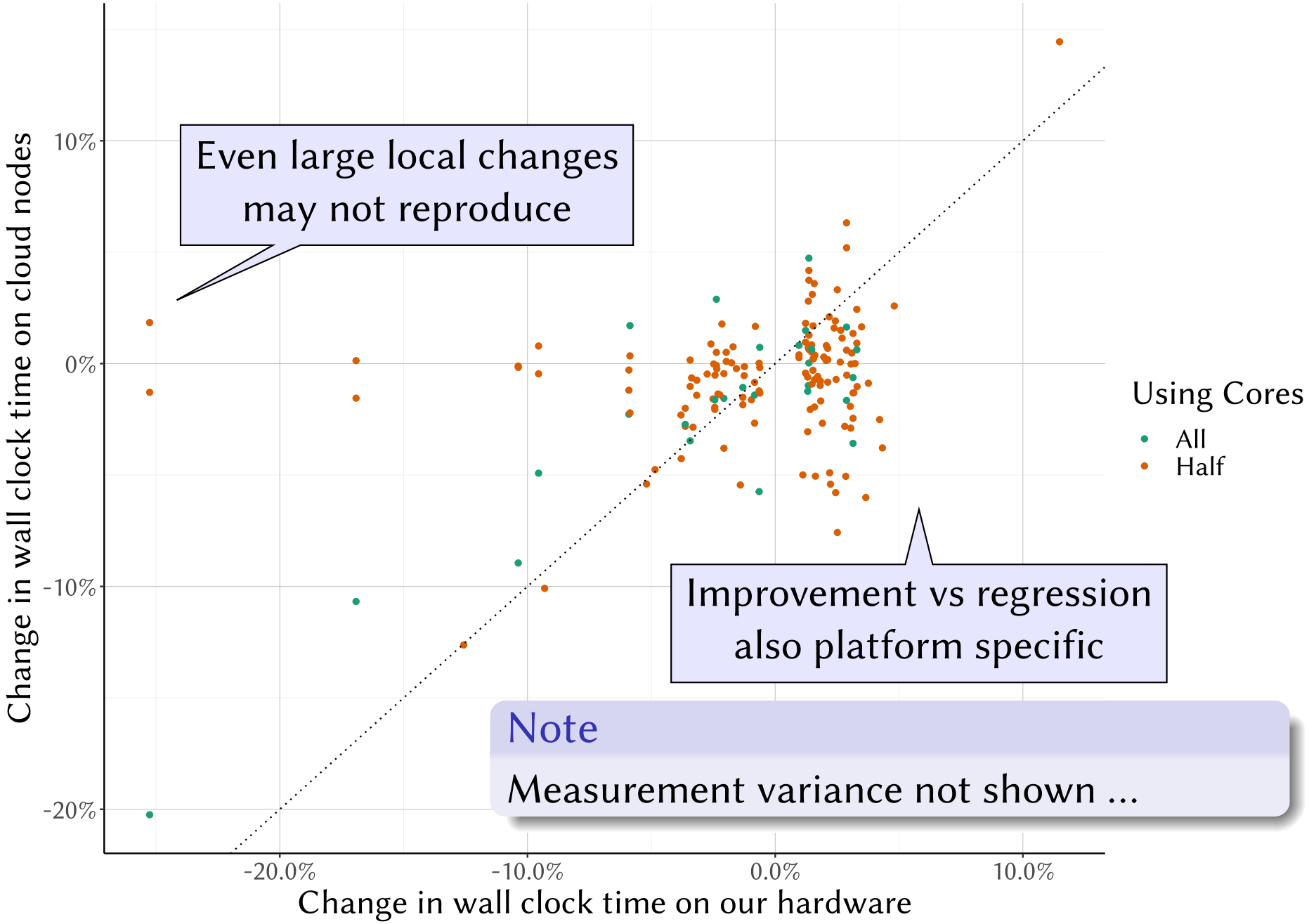
Wall Clock Time Changes Not Always Portable



Wall Clock Time Changes Not Always Portable



Wall Clock Time Changes Not Always Portable



Note

Measurement variance not shown ...

Take Away So Far ...

Looking at more execution metrics can improve accuracy

- Can help developers trust detected time changes
- Or even direct investigation of change causes

Not really clear how to combine multiple (possibly) conflicting results

- Some metrics changing and some not
- Some platforms improving and some regressing
- Some benchmarks improving and some regressing

Outline

Is Historical Data Useful ?

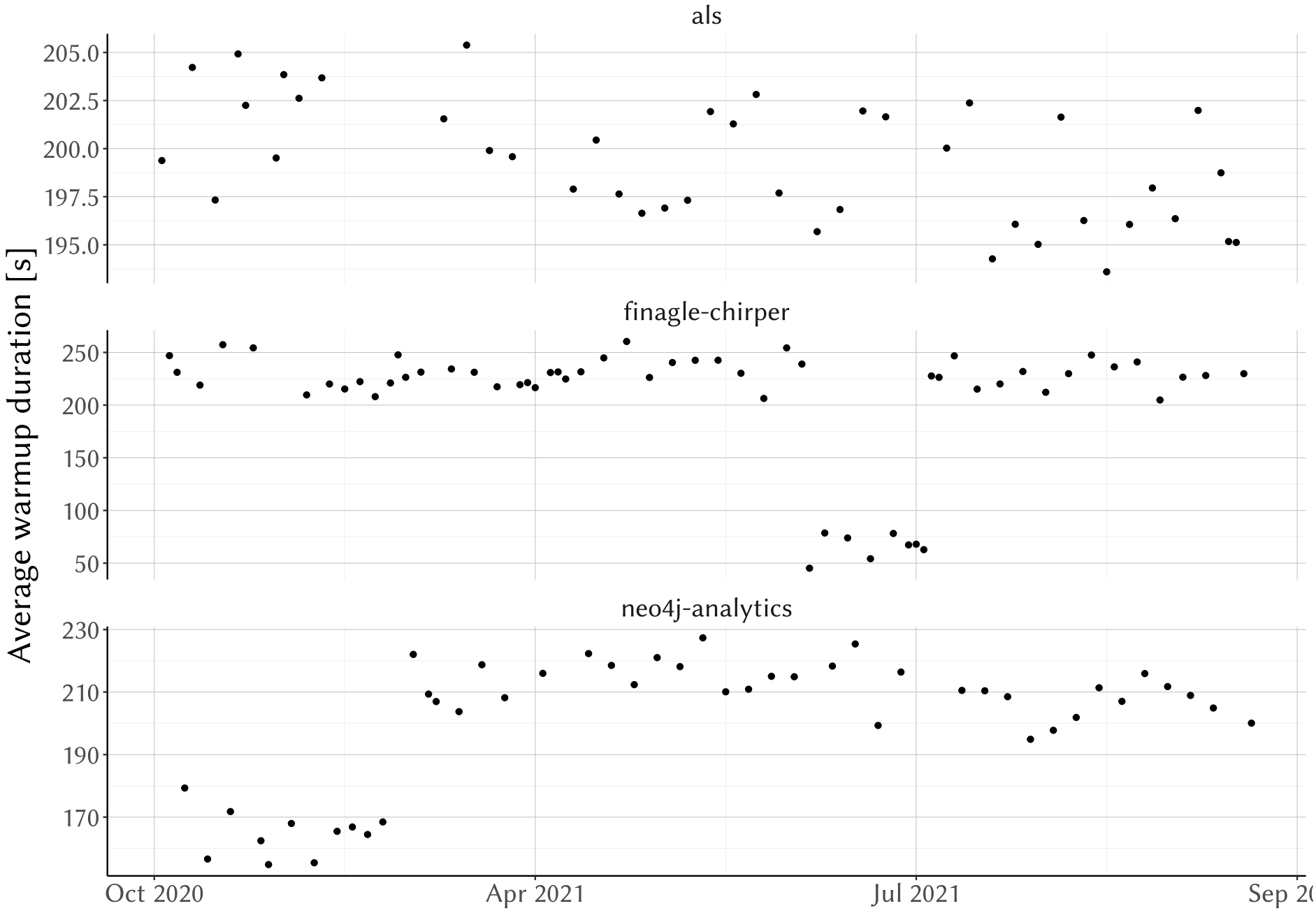
What use for historical measurements ?

- Not in time series analysis
  - Old history not necessarily relevant
  - Enough data in recent measurements
- But other system properties may prove stable
- This can help with warm up and accuracy computations

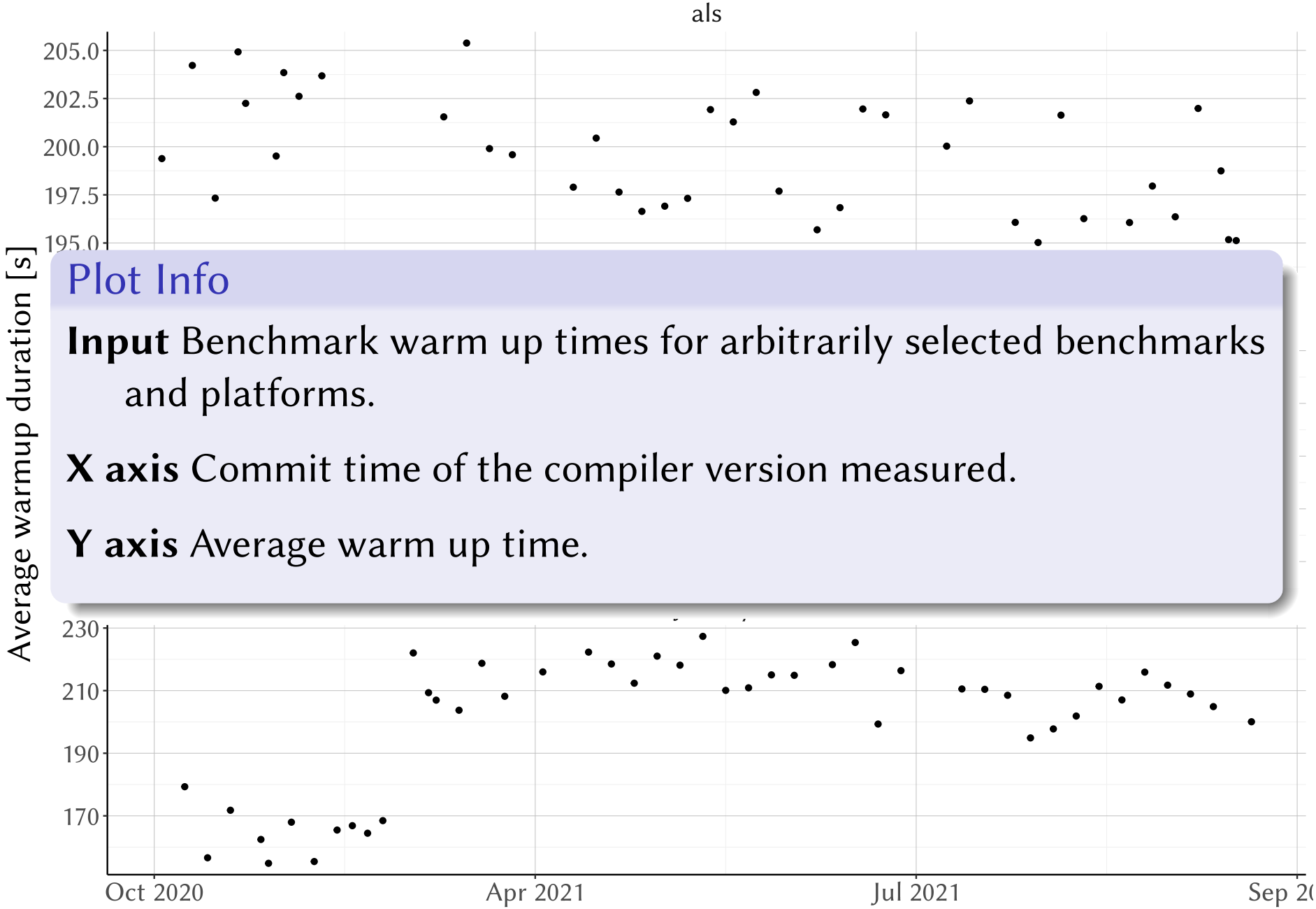
What we plan

- To reduce too long warm up
  - Use recent warm up history to set measurement duration
  - Occasional long measurements to probe for warm up changes
- To reduce too high run count
  - Use recent accuracy history to set run count
  - Incremental measurements to avoid false negatives

Warm Up History Per Benchmark



Warm Up History Per Benchmark



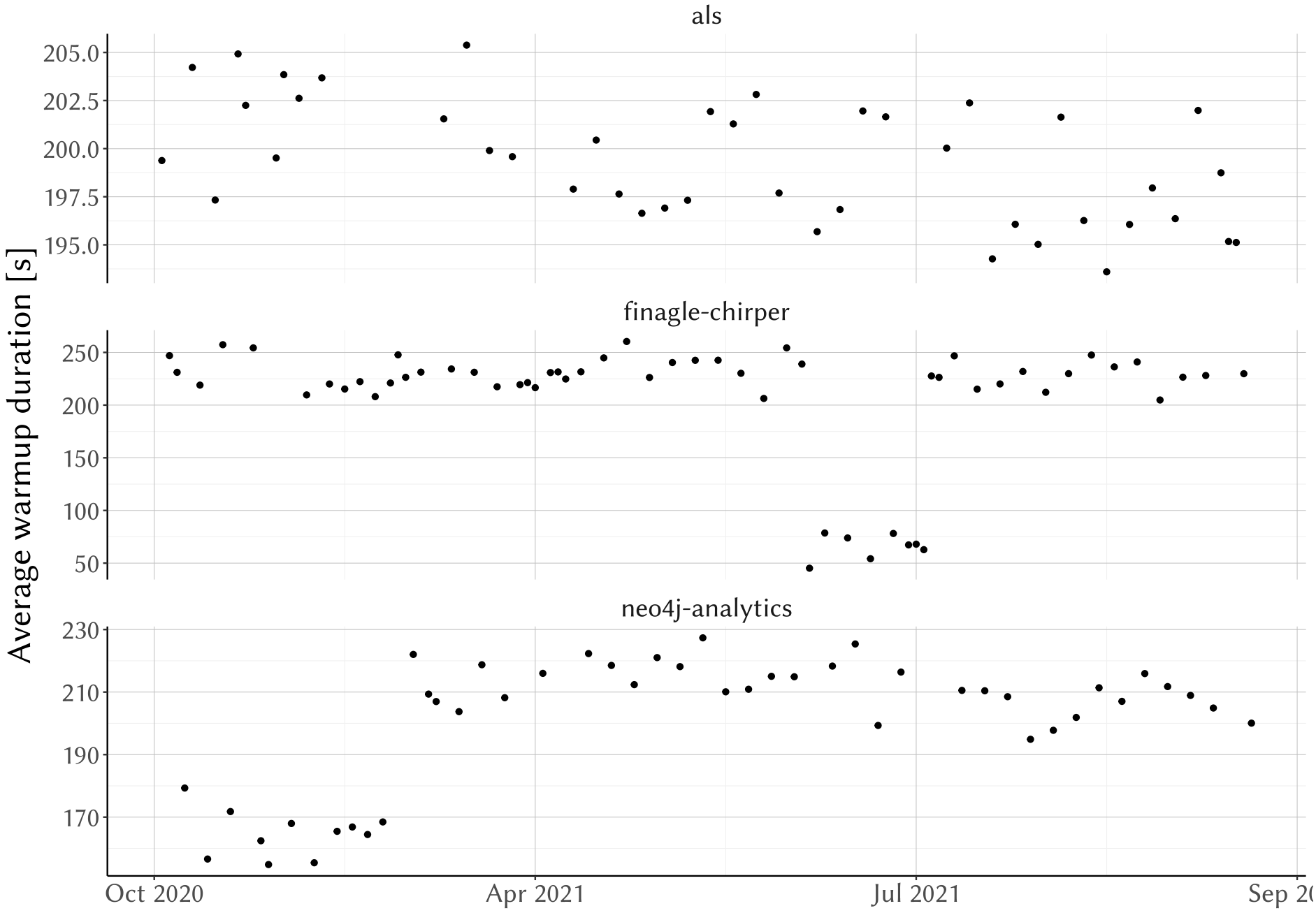
Plot Info

Input Benchmark warm up times for arbitrarily selected benchmarks and platforms.

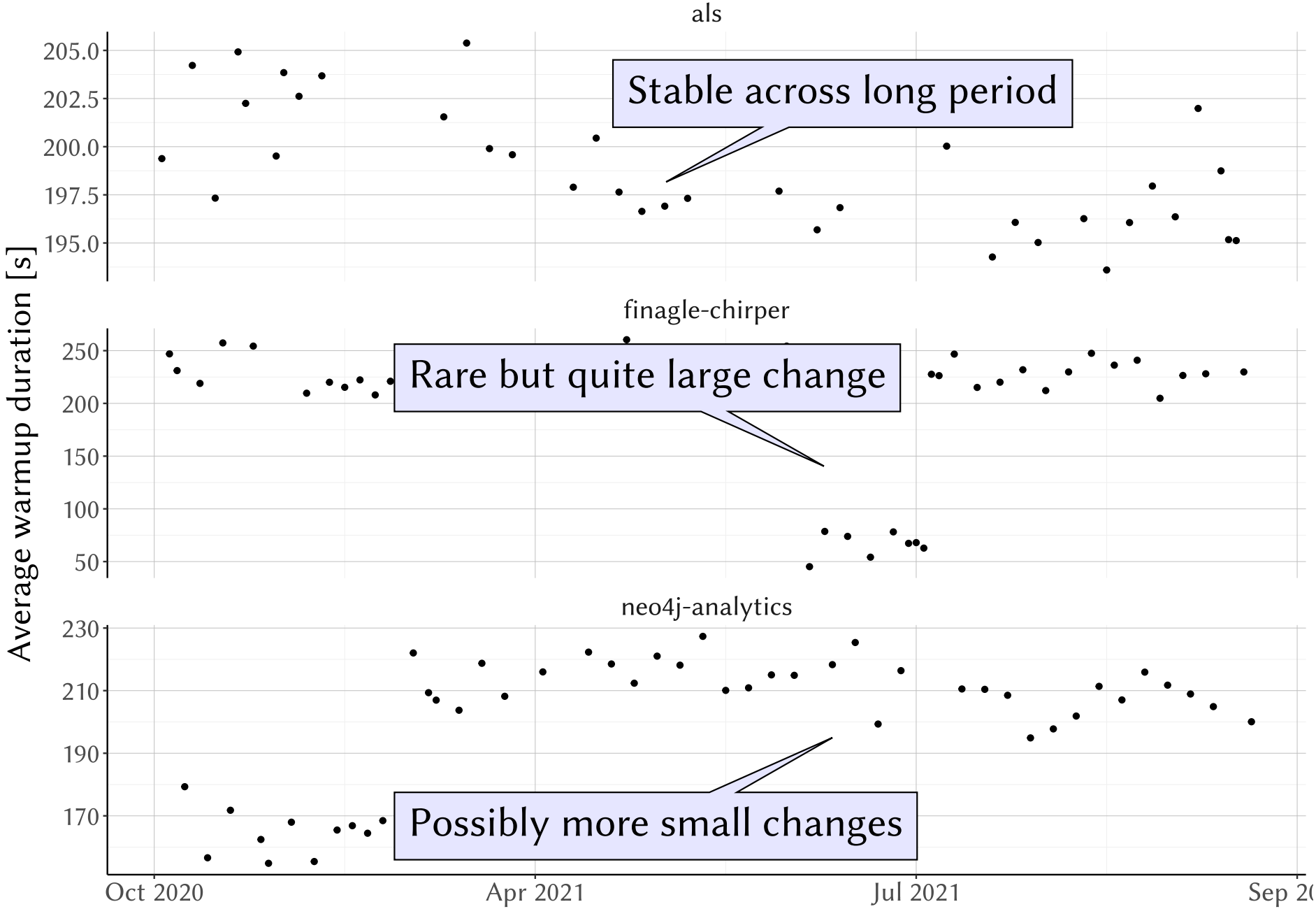
X axis Commit time of the compiler version measured.

Y axis Average warm up time.

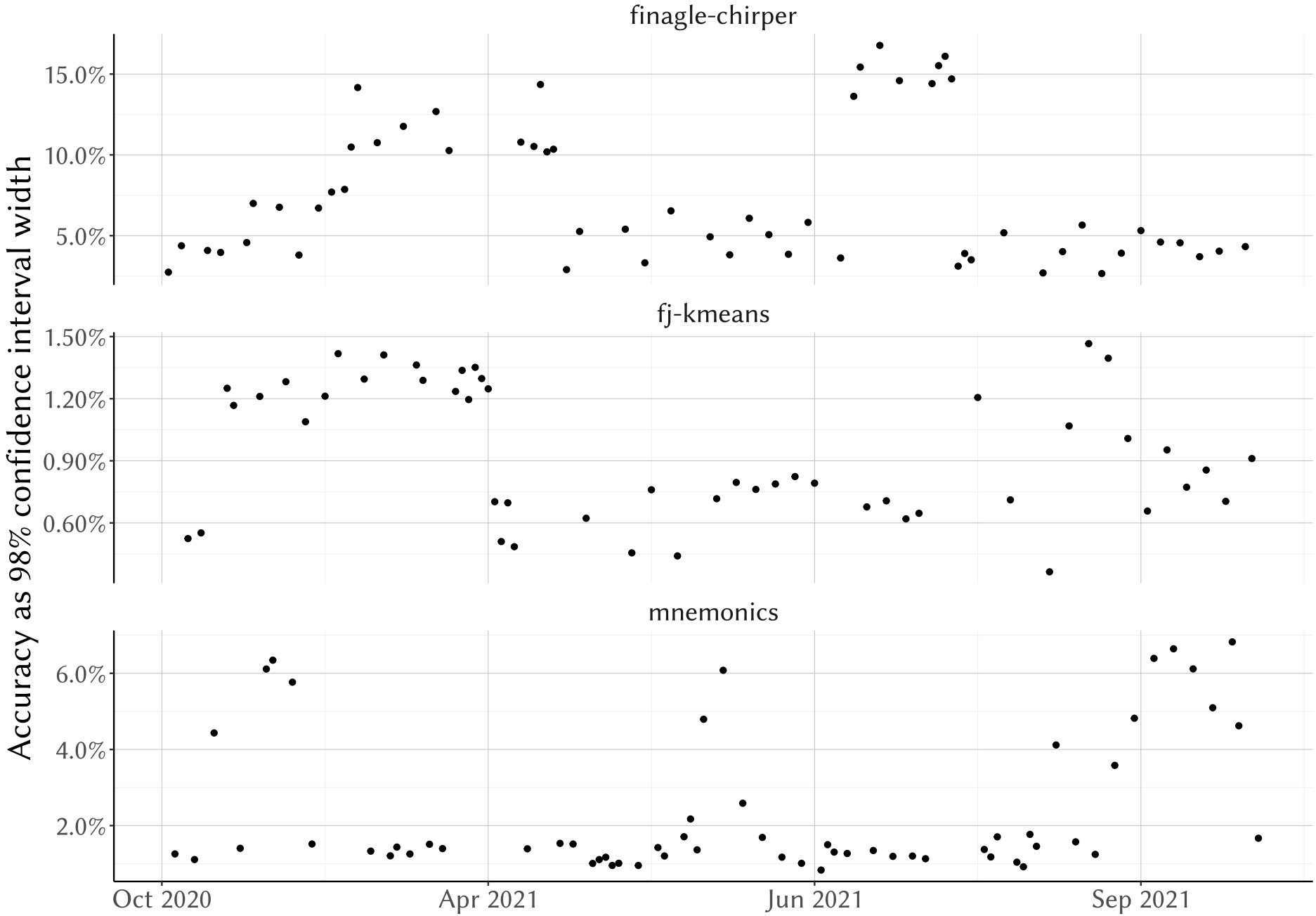
Warm Up History Per Benchmark



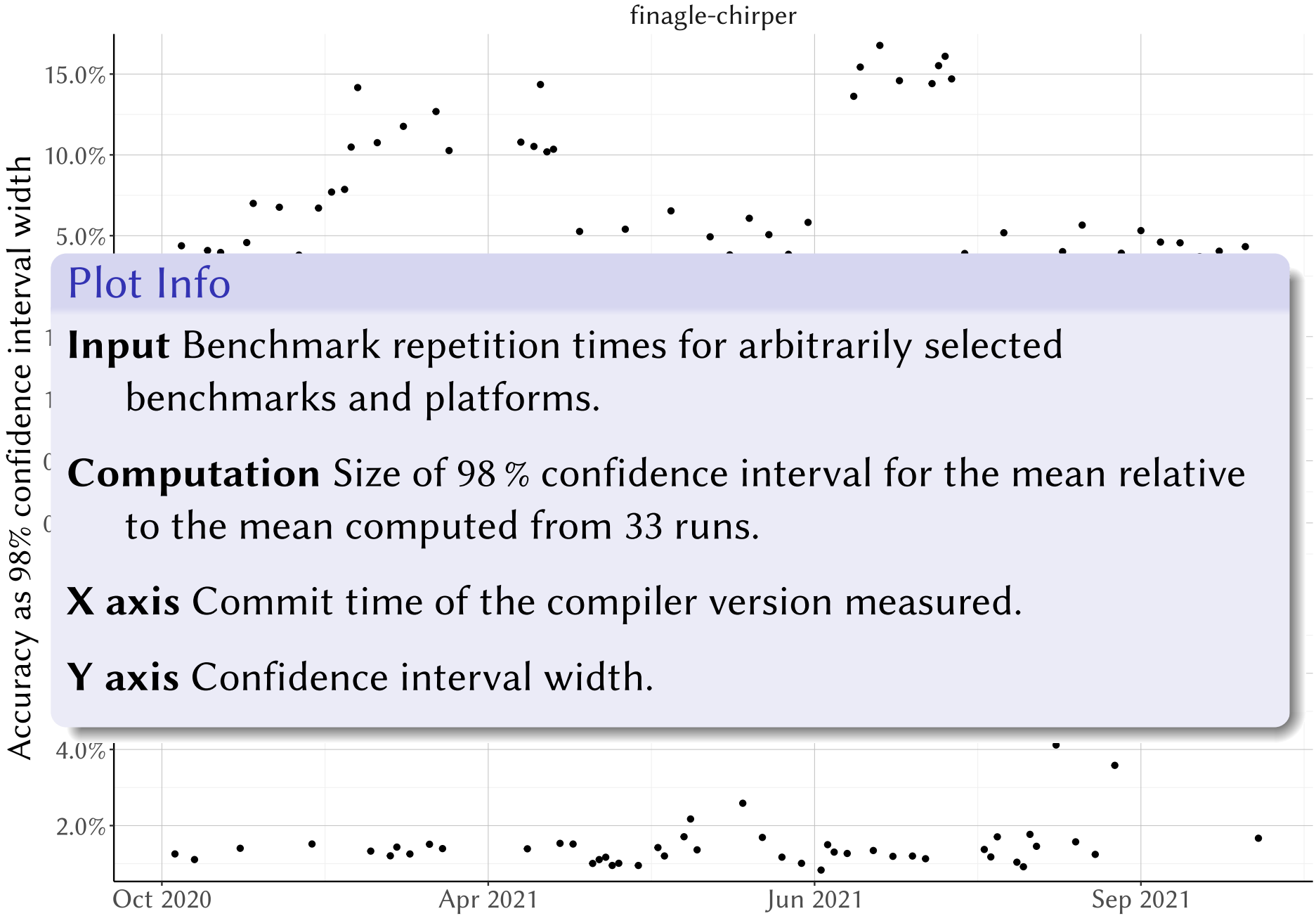
Warm Up History Per Benchmark



Accuracy History Per Benchmark



Accuracy History Per Benchmark



Plot Info

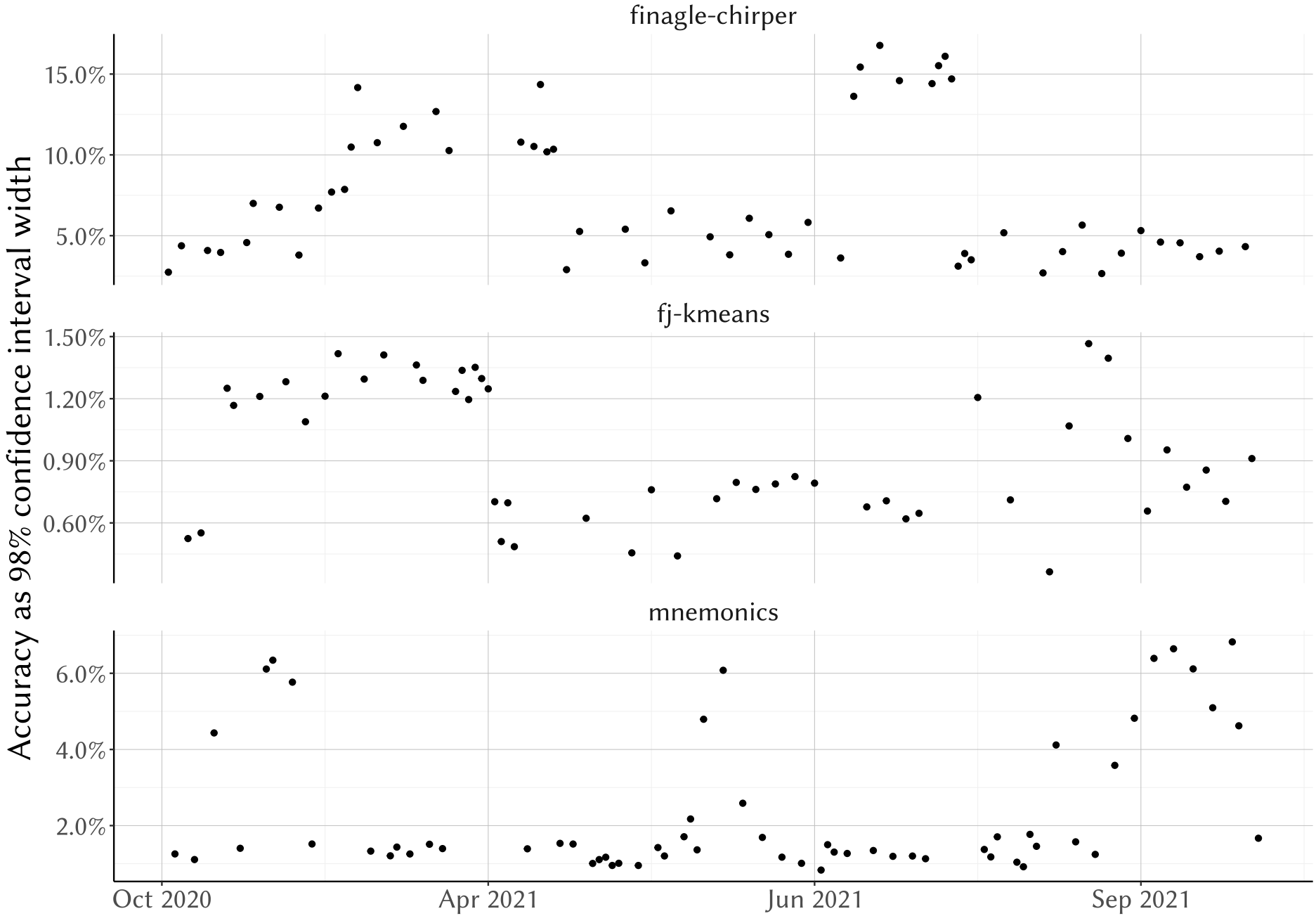
Input Benchmark repetition times for arbitrarily selected benchmarks and platforms.

Computation Size of 98 % confidence interval for the mean relative to the mean computed from 33 runs.

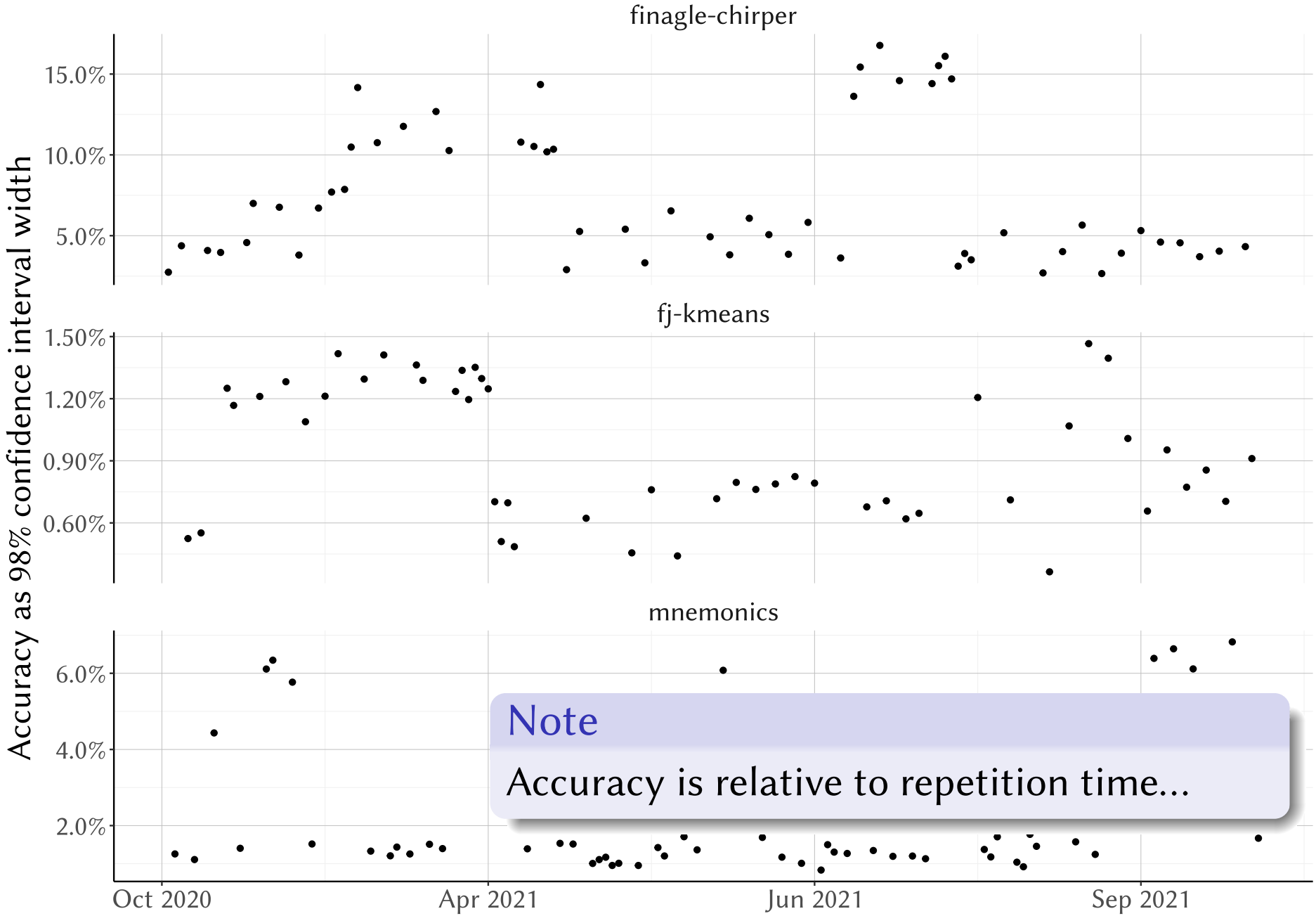
X axis Commit time of the compiler version measured.

Y axis Confidence interval width.

Accuracy History Per Benchmark



Accuracy History Per Benchmark



Note

Accuracy is relative to repetition time...

Take Away So Far ...

Some measurement properties exhibit useful stability across history

- Warm up duration
- Benchmark accuracy

Not yet clear how to use this information in automated measurements

Outline

Manual Change Classification

We used manual classification to assess functionality

- Ad hoc selection of compiler version intervals
- Benchmarks not necessarily represented equally
- More measurements added when not sure

We have no classification information about false negatives

- Likely impacts especially small changes relative to variance

Manual Change Classification

We used manual classification to assess functionality

- Ad hoc selection of compiler version intervals
- Benchmarks not necessarily represented equally
- More measurements added when not sure

We have no classification information about false negatives

- Likely impacts especially small changes relative to variance

Plot Info

Input Benchmark repetition times for arbitrarily selected pairs of platform versions with suspected change.

X axis Benchmark repetitions and runs ordered sequentially.

Y axis Time of single benchmark repetition.

Color Distinguishes versions.

Classification Example: Trivial

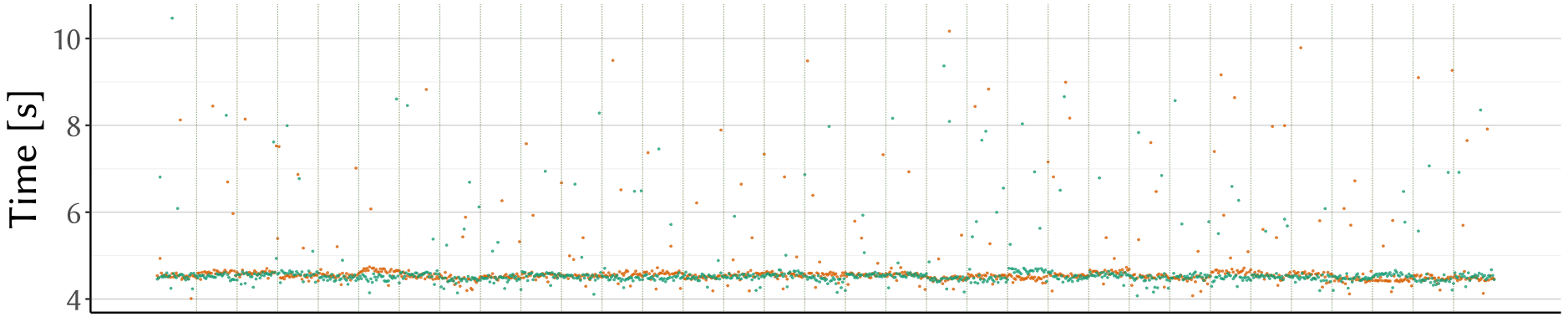
Classification Example: Trivial

An obvious difference that is trivial to classify

- Very low variance both within run and between runs
- Difference of large relative magnitude

If all data looked like this we would have little to talk about ...

Classification Example: Small Change

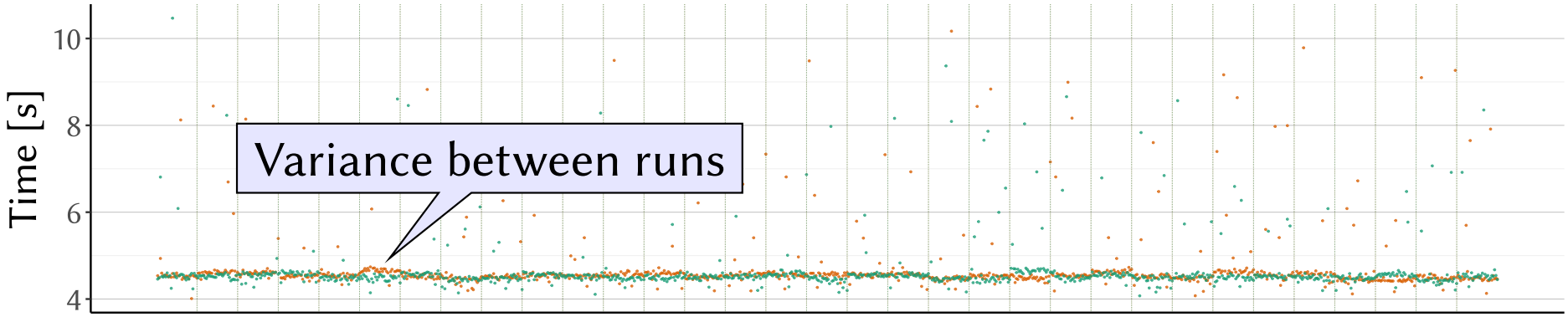


Classification Example: Small Change

Computed difference in average repetition time around 0.6 %

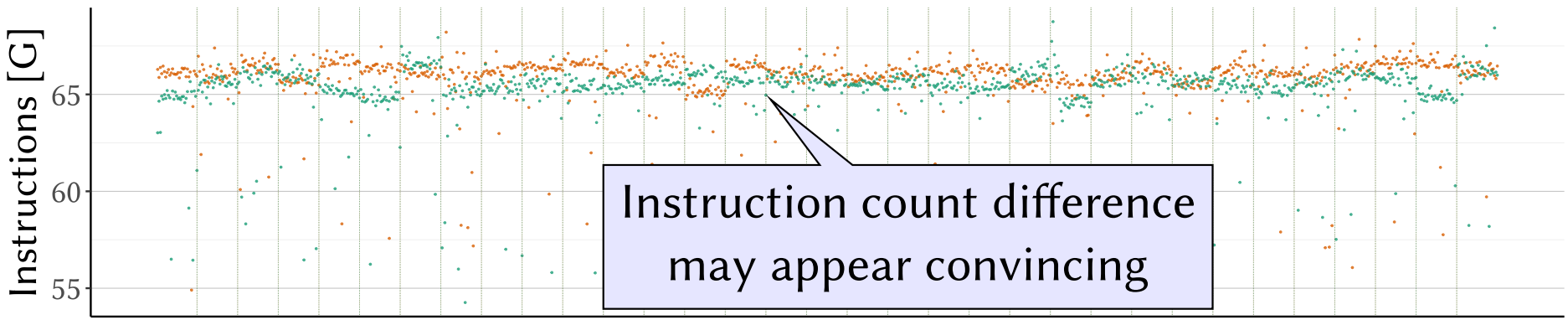
- Variance between runs large relative to the computed difference
- Outliers large relative to the computed difference
- Maybe we need more data ?

Classification Example: Small Change

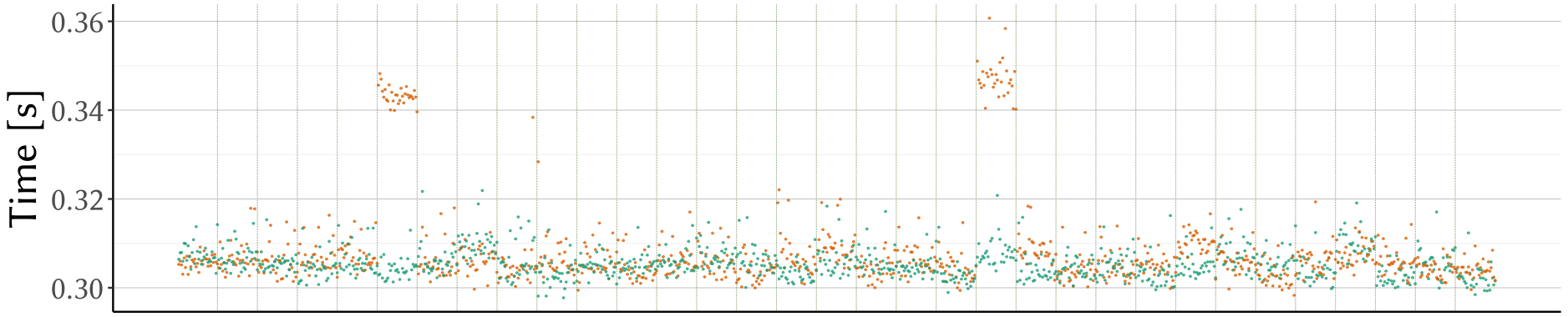


Computed difference in average repetition time around 0.6 %

- Variance between runs large relative to the computed difference
- Outliers large relative to the computed difference
- Maybe we need more data ?



Classification Example: Outlier Definition Issues

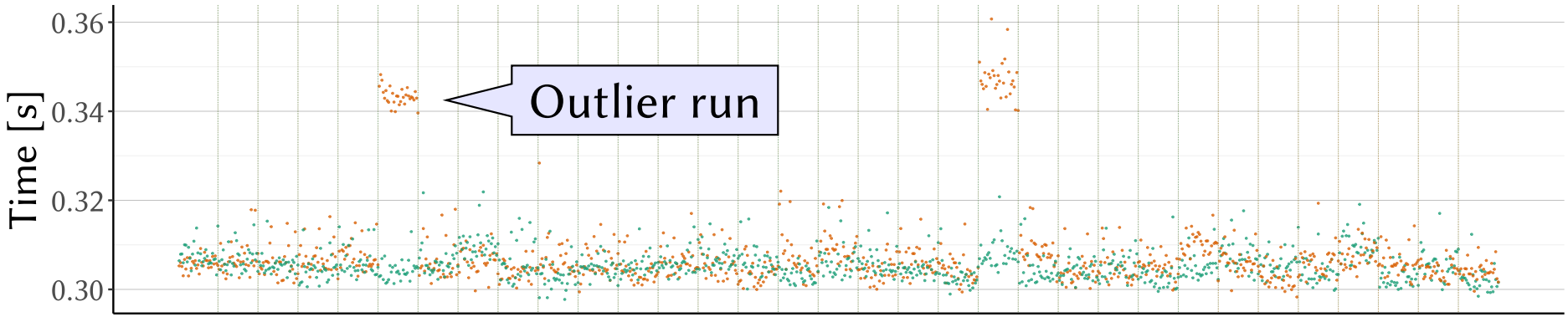


Classification Example: Outlier Definition Issues

Computed difference in average repetition time around 0.9 %

- The computed difference very much depends on outlier filtering
- Are we sure we have enough data ?

Classification Example: Outlier Definition Issues



Computed difference in average repetition time around 0.9 %

- The computed difference very much depends on outlier filtering
- Are we sure we have enough data ?

Assume 10 % change in outlier runs and 10 % chance of such runs

- This would result in an average repetition time change of 0.9 %
- There is around 35 % chance of getting 10 fine runs
- Obviously the example can be stretched in various directions

Useless Change Reports ?

Typical scenarios assume cause-and-effect relationship between commit and performance change

- Commit introduces reason for performance change
- Performance change observed in measurements
- Change can be undone by reverting commit

But what if the situation is more complicated ?

- Change caused by commit but commit not responsible for change
- Change impacting different benchmarks in different ways
- Change impacting different platforms in different ways
- Change expected but need to assess cost vs benefit
- Change seen on benchmark believed artificial
- Change impacts outliers
- ...

Regression Example: Processor Scheduling I

Code

A microbenchmark that locates the first negative array item.

```
def run () {
    for (i <- 0 until REPEATS) {
        blackhole += findNegative (numbers)
    }
}

def findNegative (numbers: Array[Int]): Option[Int] = {
    numbers.find(_ < 0)
}
```

What the measurements said

Clear repetition time change between roughly 230 ms and roughly 170 ms

No change in other observed counters like instruction count

Observed multiple times in versions across several days

Commit changes often clearly unrelated

Regression Example: Processor Scheduling II

Assembly

Compilation results in reasonably compact assembly code.

```
0x00007f115c894c00: cmp    %r13d,%edi                ;loop iteration count test
0x00007f115c894c03: jbe    0x00007f115c89561c
0x00007f115c894c09: mov    0x10(%rdx,%r13,4),%r10d   ;fetch array item
0x00007f115c894c0e: test   %r10d,%r10d               ;negative test
0x00007f115c894c11: jl     0x00007f115c894c2a        ;found negative
0x00007f115c894c17: test   %eax,0x1942d3e9(%rip)     ;safepoint poll
0x00007f115c894c1d: inc    %r13d
0x00007f115c894c20: cmp    %r13d,%edi                ;loop iteration count test (again)
0x00007f115c894c23: jg     0x00007f115c894c00
```

Analysis

Inner loop executes at IPC 6 when fast or IPC 4.5 when slow
Performance difference inflated from mere 0.5 cycle per iteration
Instruction scheduler counters report different µops port use as the reason
Actual scheduler choice only indirectly influenced by code

Regression Example: Inlining Heuristic I

Code

A microbenchmark that filters odd array items.

```
def run () {
    for (i <- 0 until REPEATS) {
        blackhole += filterOdd (numbers).length
    }
}

def filterOdd (numbers: ArrayBuffer[Int]): ArrayBuffer[Int] = {
    numbers.filter (_ % 2 == 1)
}
```

What the measurements said

Times always stable within each run

Repetition time of a run flipping between 5 s and 5.6 s

Rarely observed runs with repetition times of roughly 3.4 s

Share of runs with each time sometimes changes between versions

Regression Example: Inlining Heuristic II

Analysis

Fast and slow runs differed in what code gets inlined
Inlining heuristic (also) relies on low level graph size of the callee

- If callee previously compiled, a cached value was used
- If callee not yet compiled, an estimate was made

Caller and callee invocation counters necessarily similar
Hence compilation jobs launched close together in time
That increases the likelihood of the inliner flipping

Take Away So Far ...

Reasons for performance change
not always directly connected to committed code

- Especially microbenchmarks may exhibit fragile performance
- Responsibility for addressing changes therefore not clear

Hard to tell what performance regressions should be addressed

- Especially with benchmarks that
  do not represent application performance
- Effort needed to investigate reasons is not very predictable
- Not clear what to do with small regressions

Thank You !

https://renaissance.dev
https://d3s.mff.cuni.cz
https://graal.d3s.mff.cuni.cz

Our work is kindly sponsored by Oracle Labs.

Thank You !

Interested in our data ?

... most data CC-BY, we also have an API

`https://renaissance.dev`
`https://d3s.mff.cuni.cz`
`https://graal.d3s.mff.cuni.cz`

Our work is kindly sponsored by Oracle Labs.

Thank You !

Interested in our data ?

... most data CC-BY, we also have an API

Contribute to Renaissance

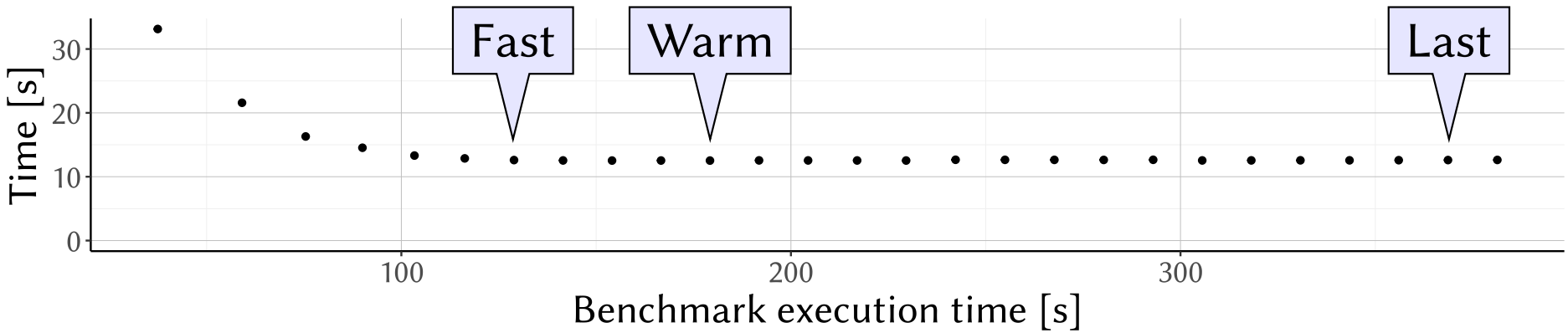
... and we will start benchmarking your code too :-)

`https://renaissance.dev`
`https://d3s.mff.cuni.cz`
`https://graal.d3s.mff.cuni.cz`

Our work is kindly sponsored by Oracle Labs.

Outline

Warm Up Enough Or Too Much ?



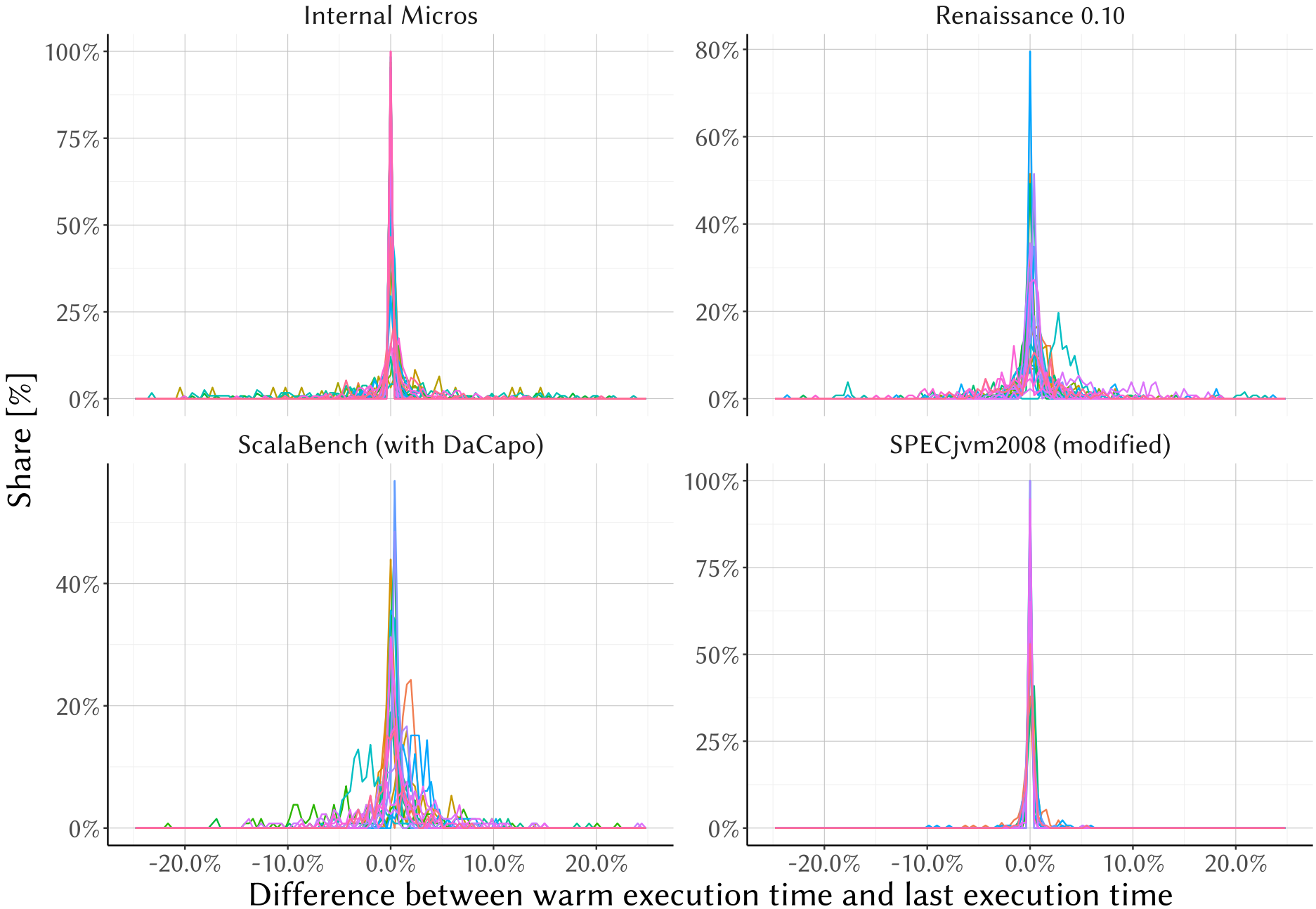
Plot Info

Fast The first repetition that is at least as fast as the warm one.

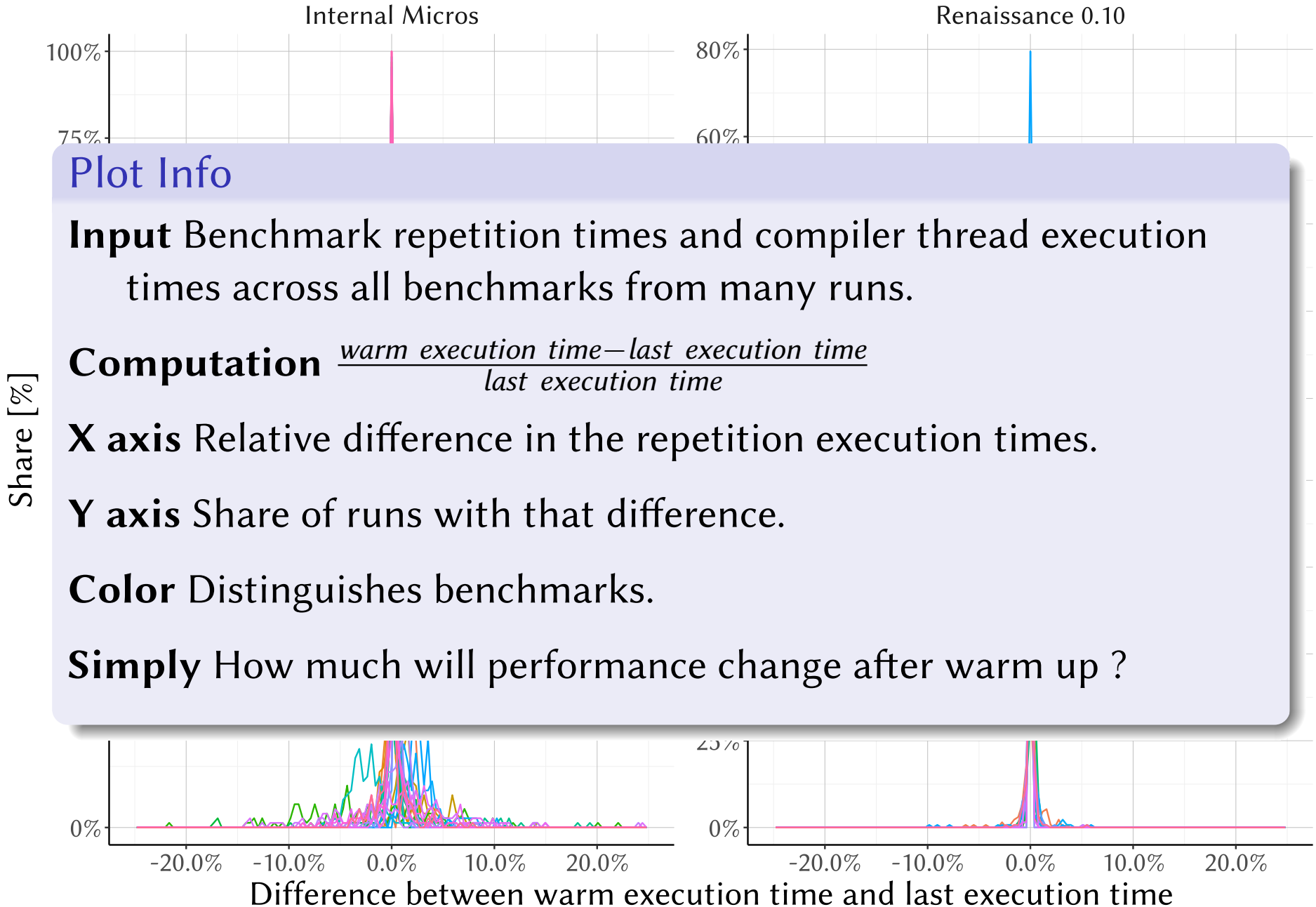
Warm The repetition that our heuristic marks as first warm.

Last The last repetition.

Do We Warm Up Enough ?



Do We Warm Up Enough ?



Plot Info

Input Benchmark repetition times and compiler thread execution times across all benchmarks from many runs.

Computation $\frac{warm\ \ execution\ \ time - last\ \ execution\ \ time}{last\ \ execution\ \ time}$

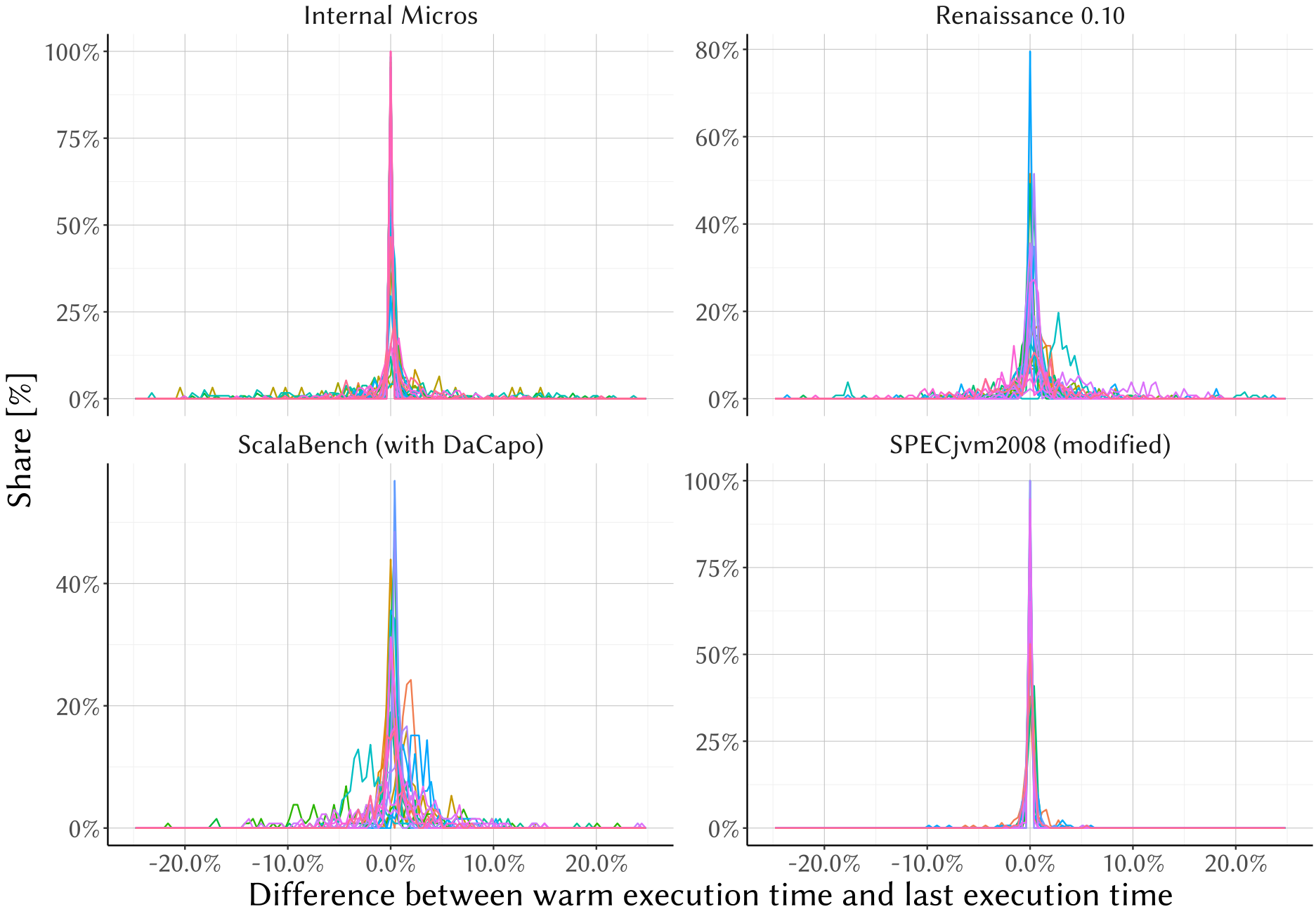
X axis Relative difference in the repetition execution times.

Y axis Share of runs with that difference.

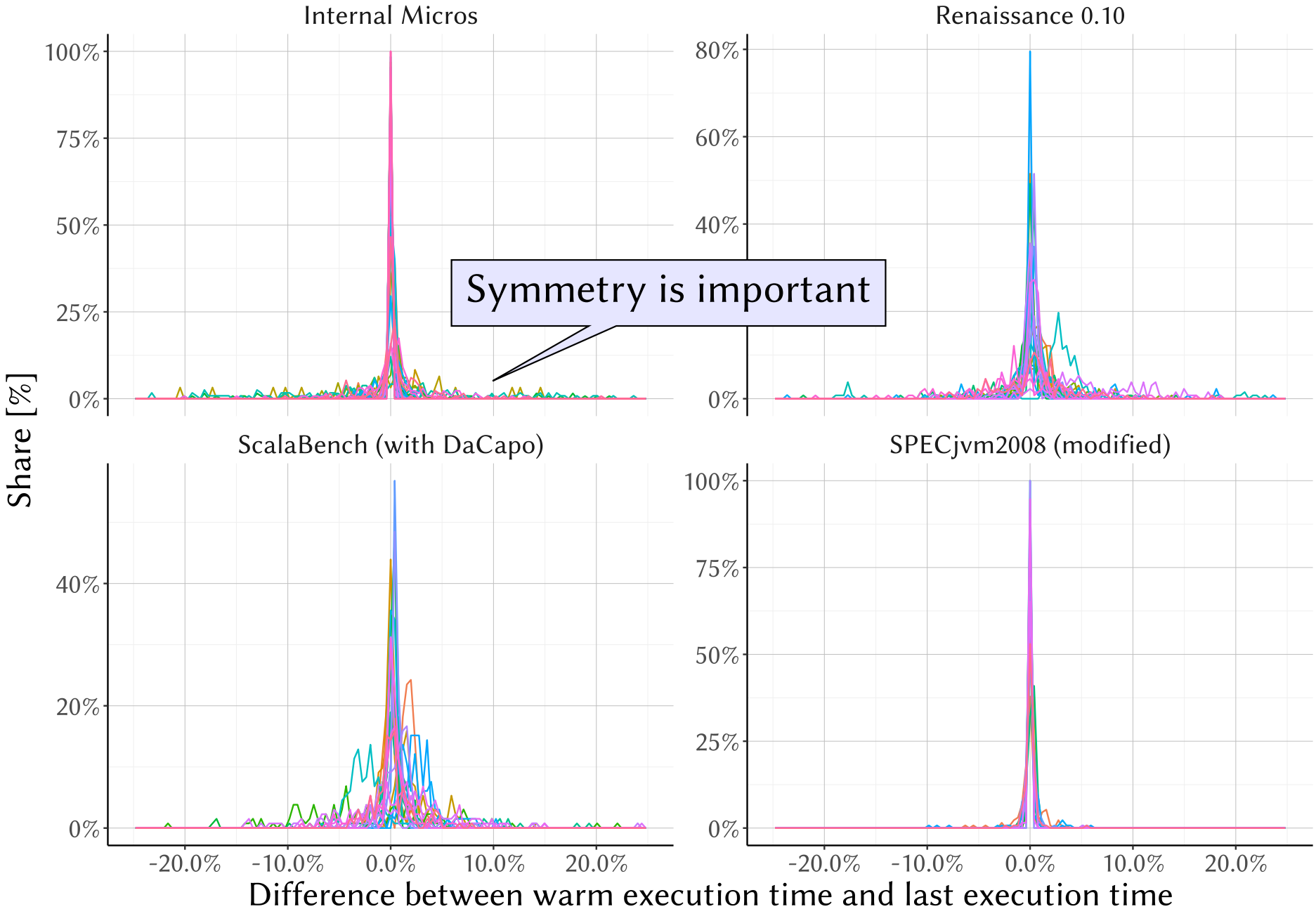
Color Distinguishes benchmarks.

Simply How much will performance change after warm up ?

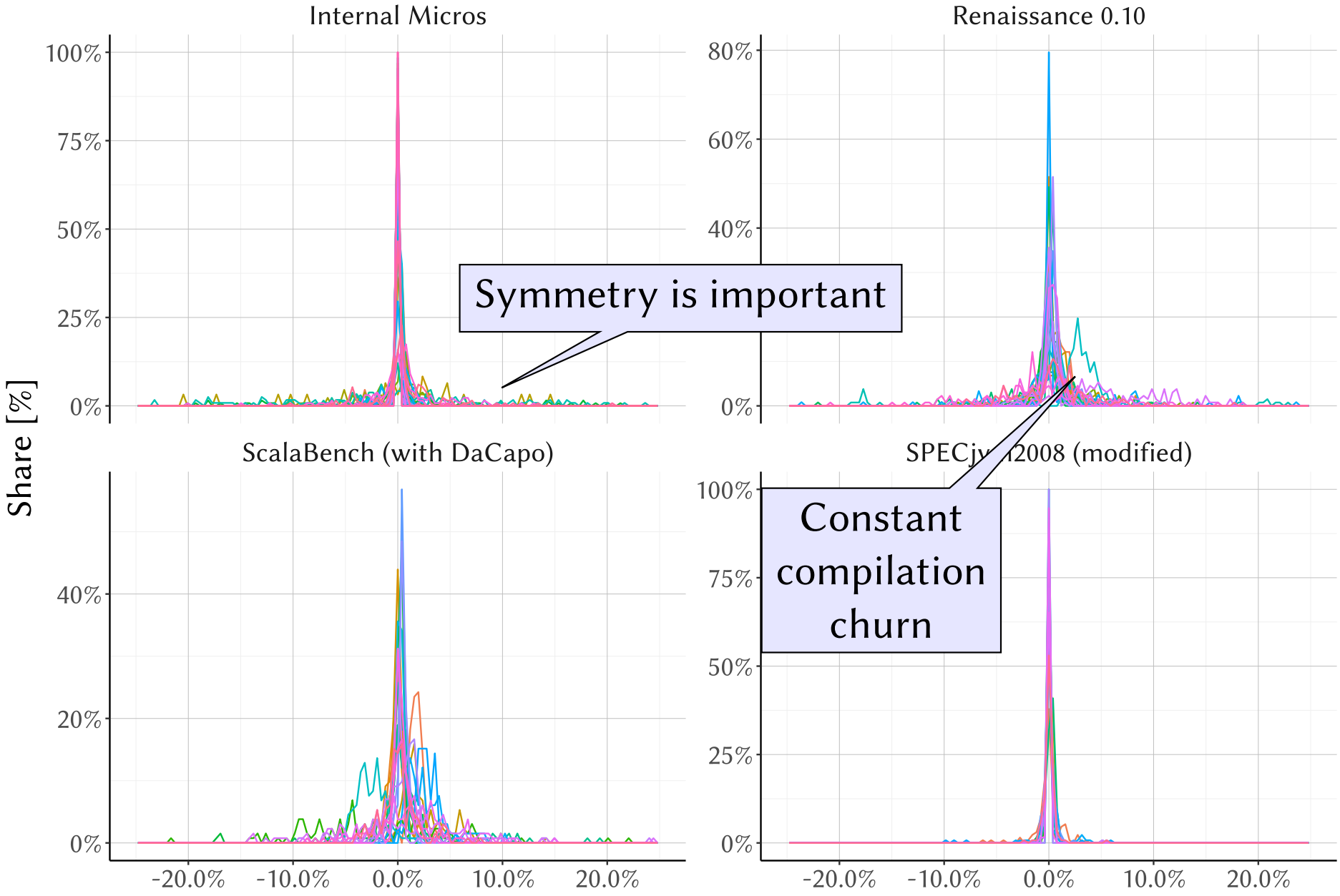
Do We Warm Up Enough ?



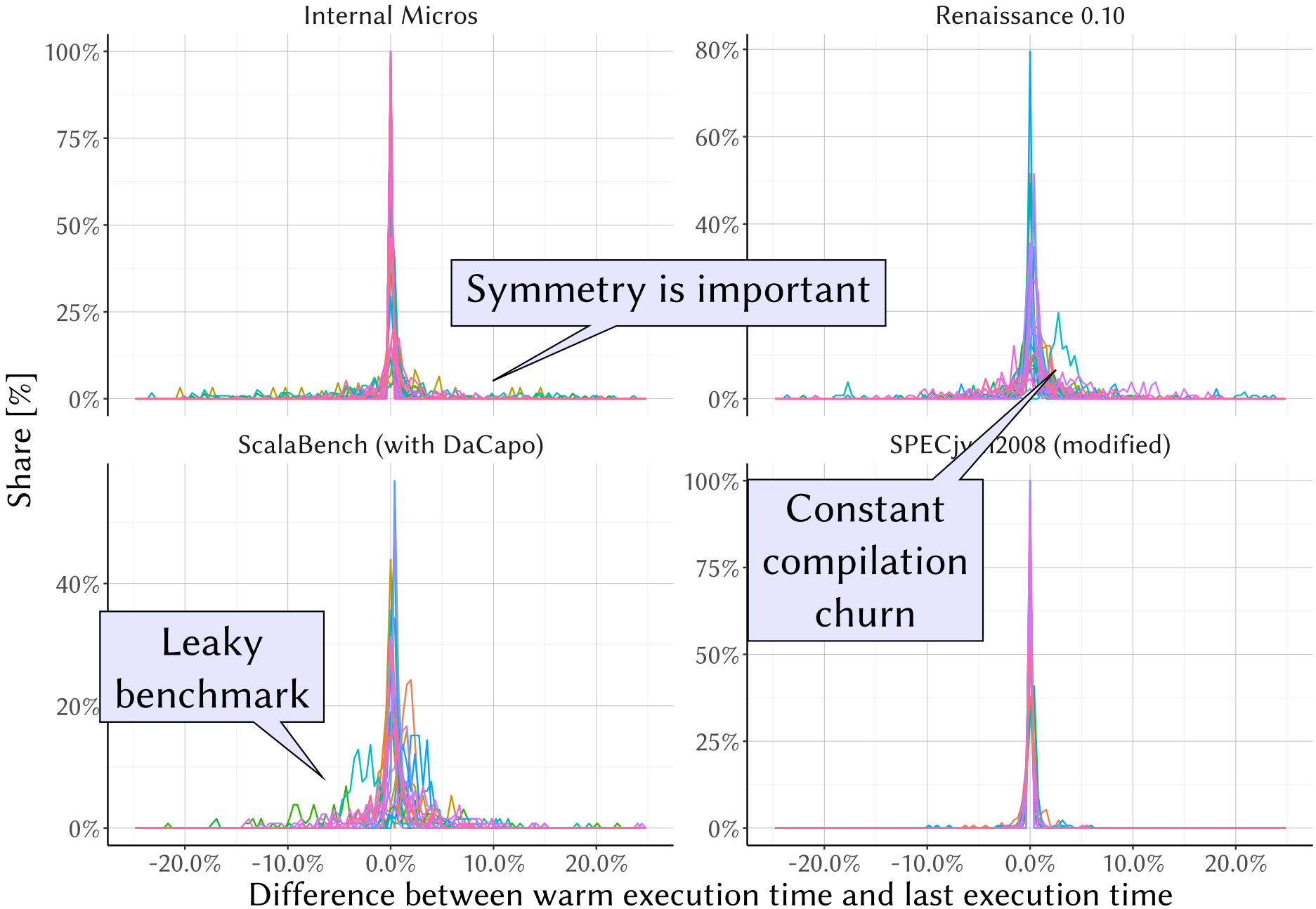
Do We Warm Up Enough ?



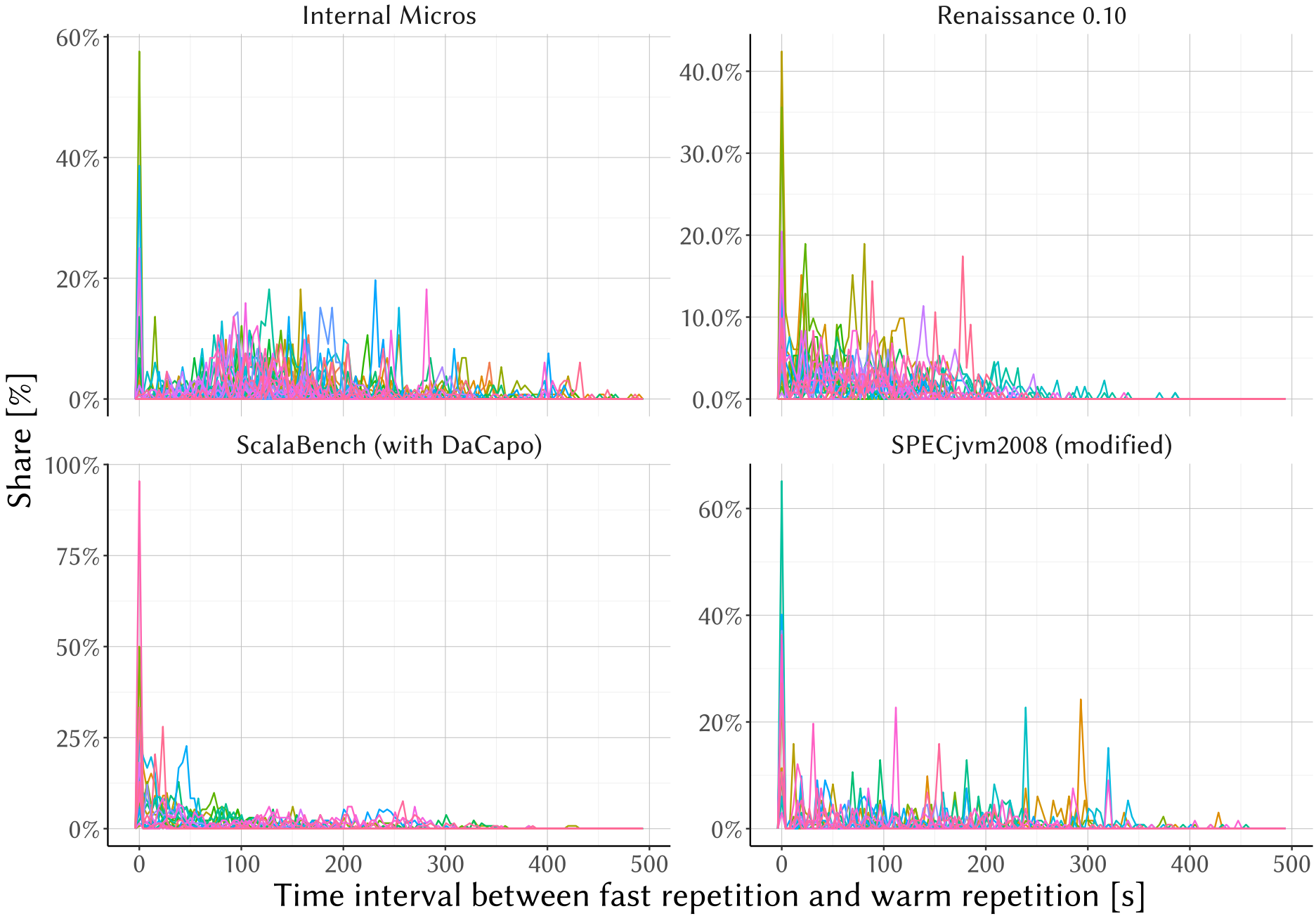
Do We Warm Up Enough ?



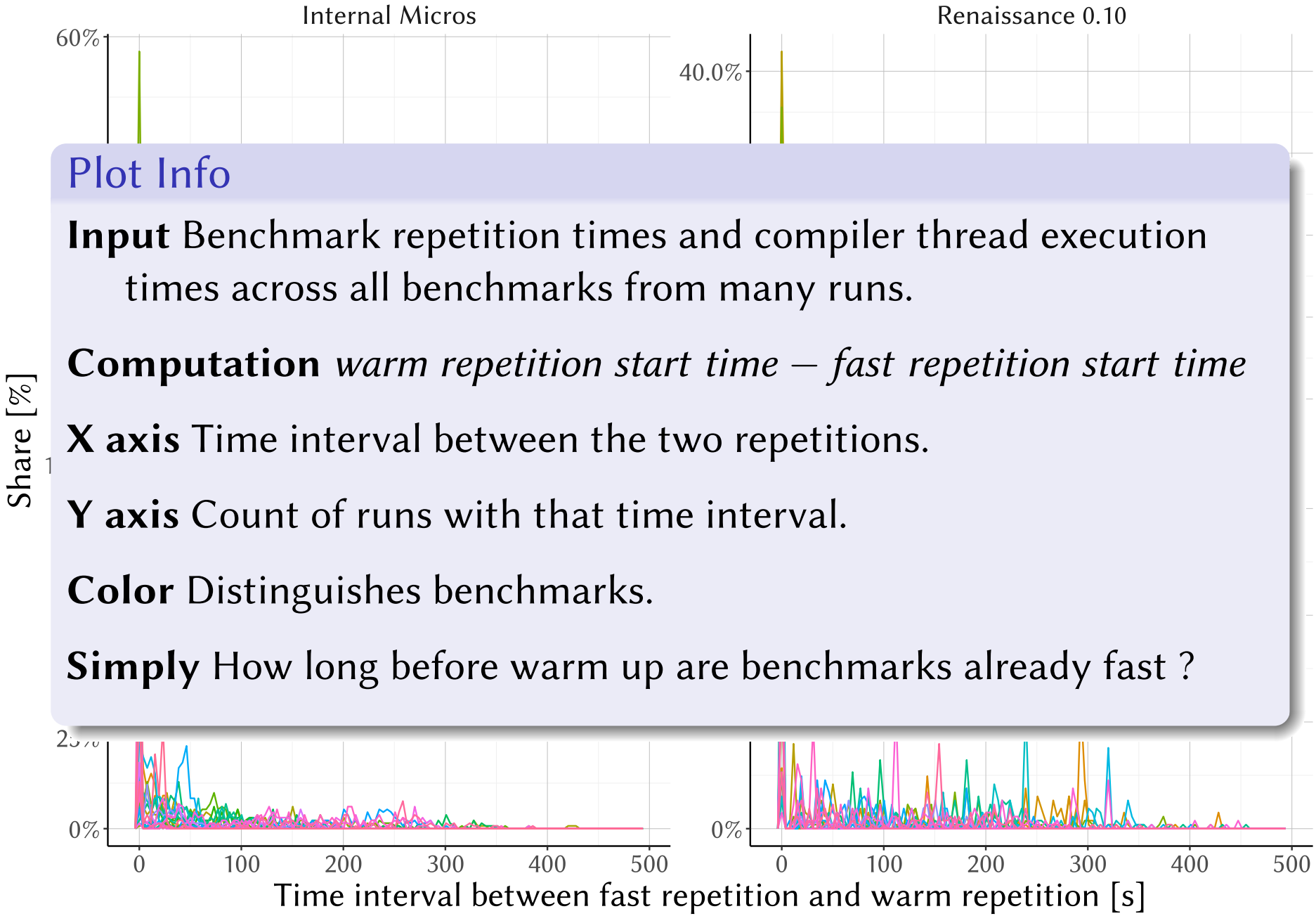
Do We Warm Up Enough ?



Do We Warm Up Too Much ?



Do We Warm Up Too Much ?



Plot Info

Input Benchmark repetition times and compiler thread execution times across all benchmarks from many runs.

Computation *warm repetition start time − fast repetition start time*

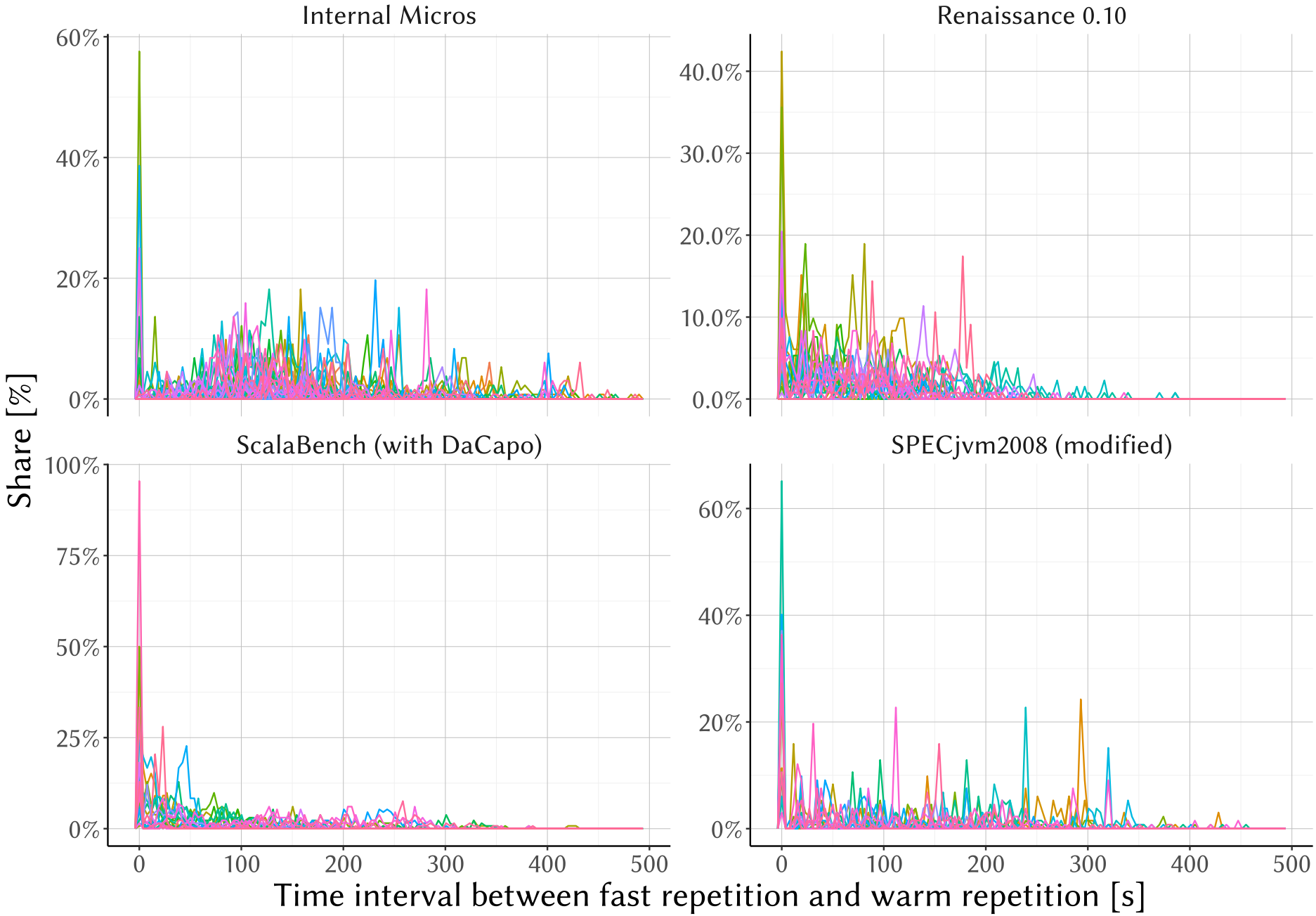
X axis Time interval between the two repetitions.

Y axis Count of runs with that time interval.

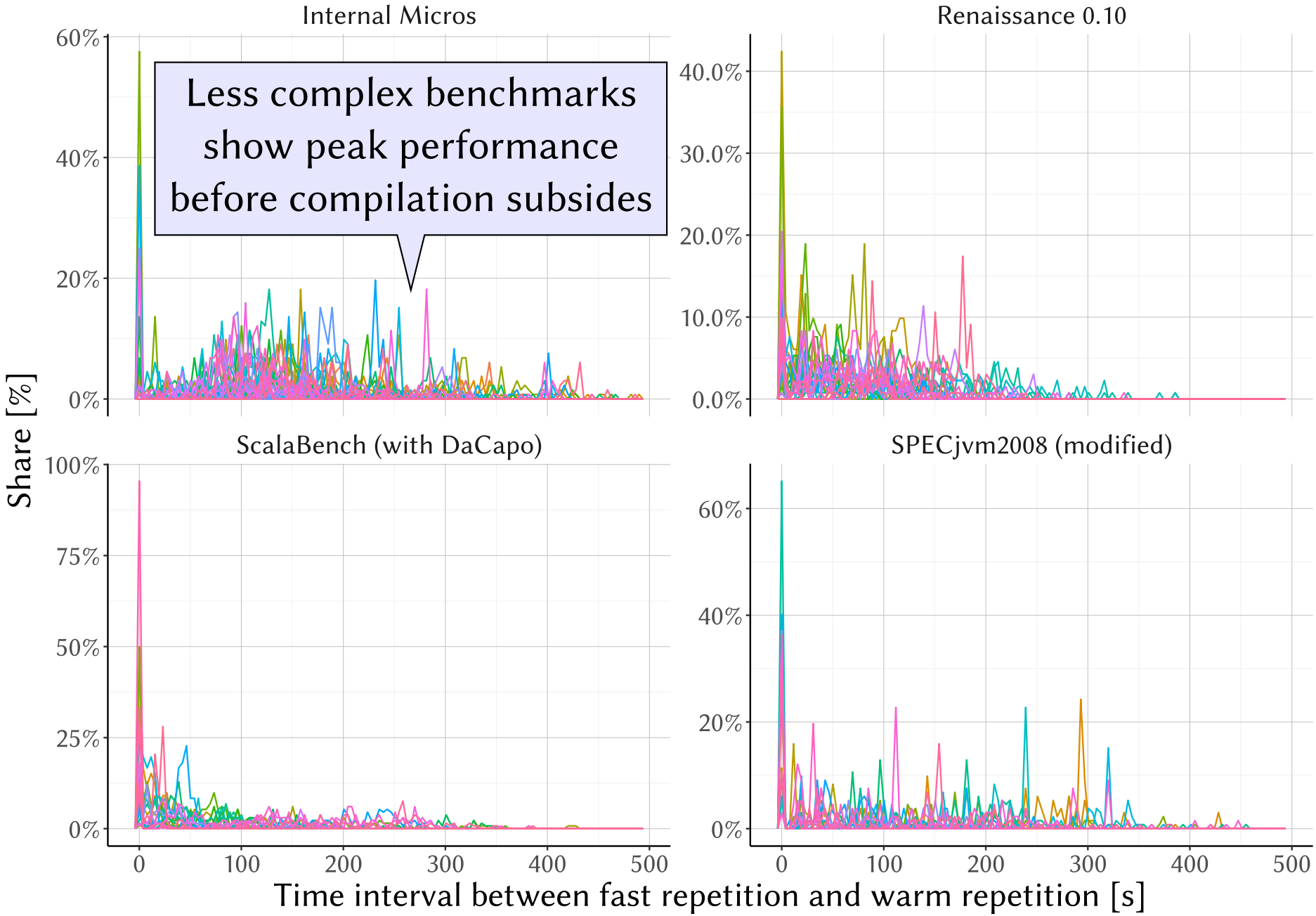
Color Distinguishes benchmarks.

Simply How long before warm up are benchmarks already fast ?

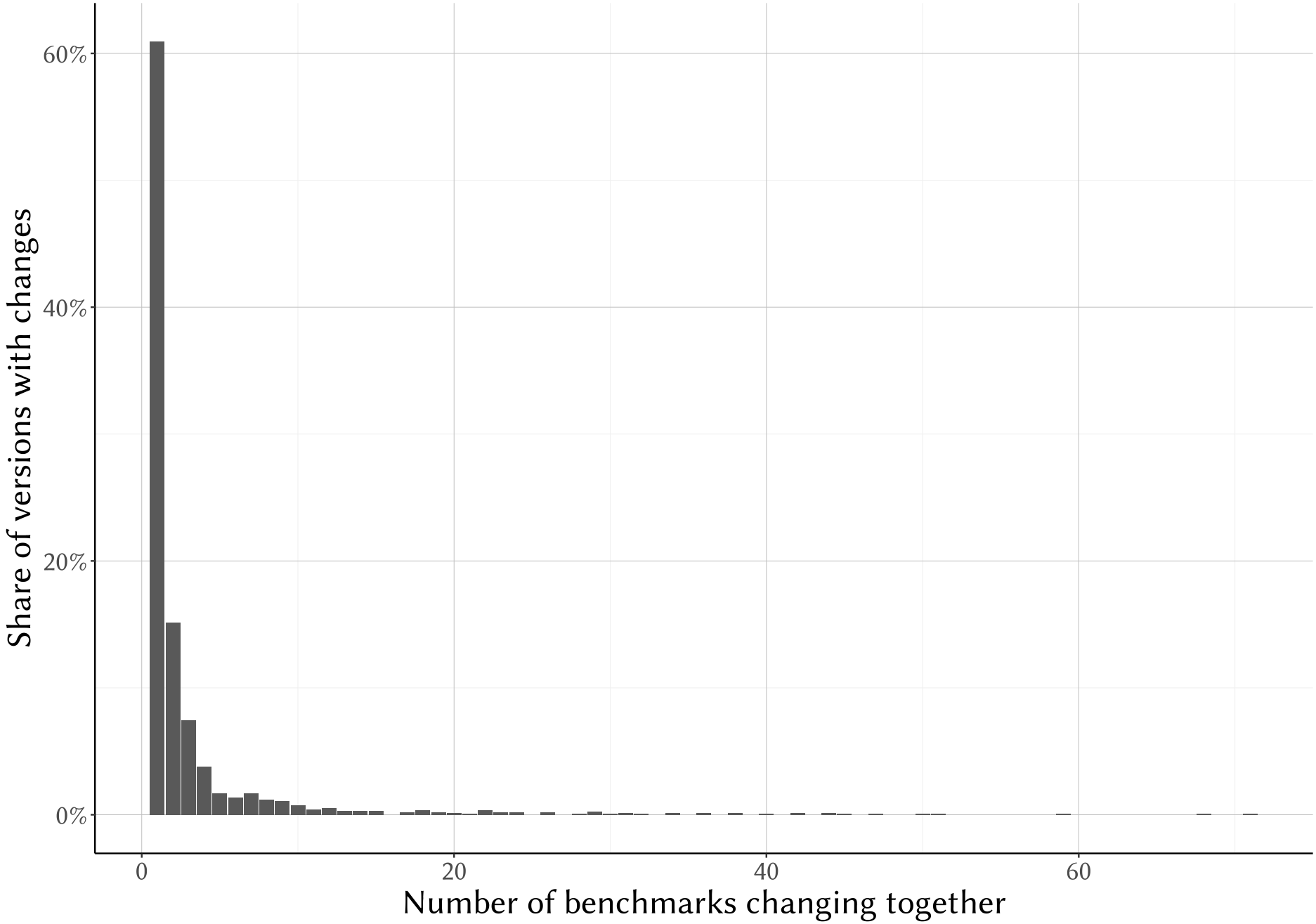
Do We Warm Up Too Much ?



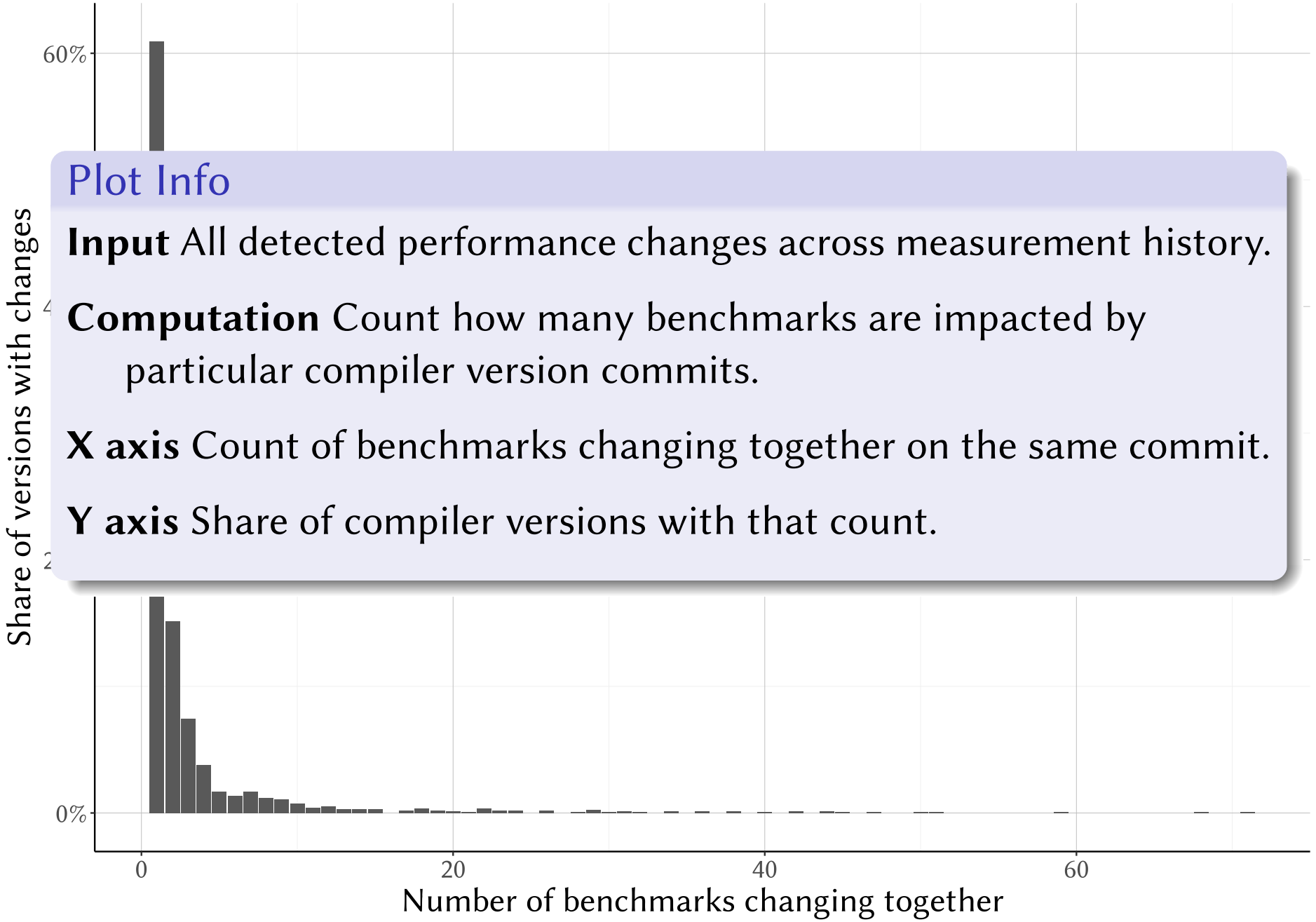
Do We Warm Up Too Much ?



Do We Have Too Many Benchmarks ?



Do We Have Too Many Benchmarks ?



Plot Info

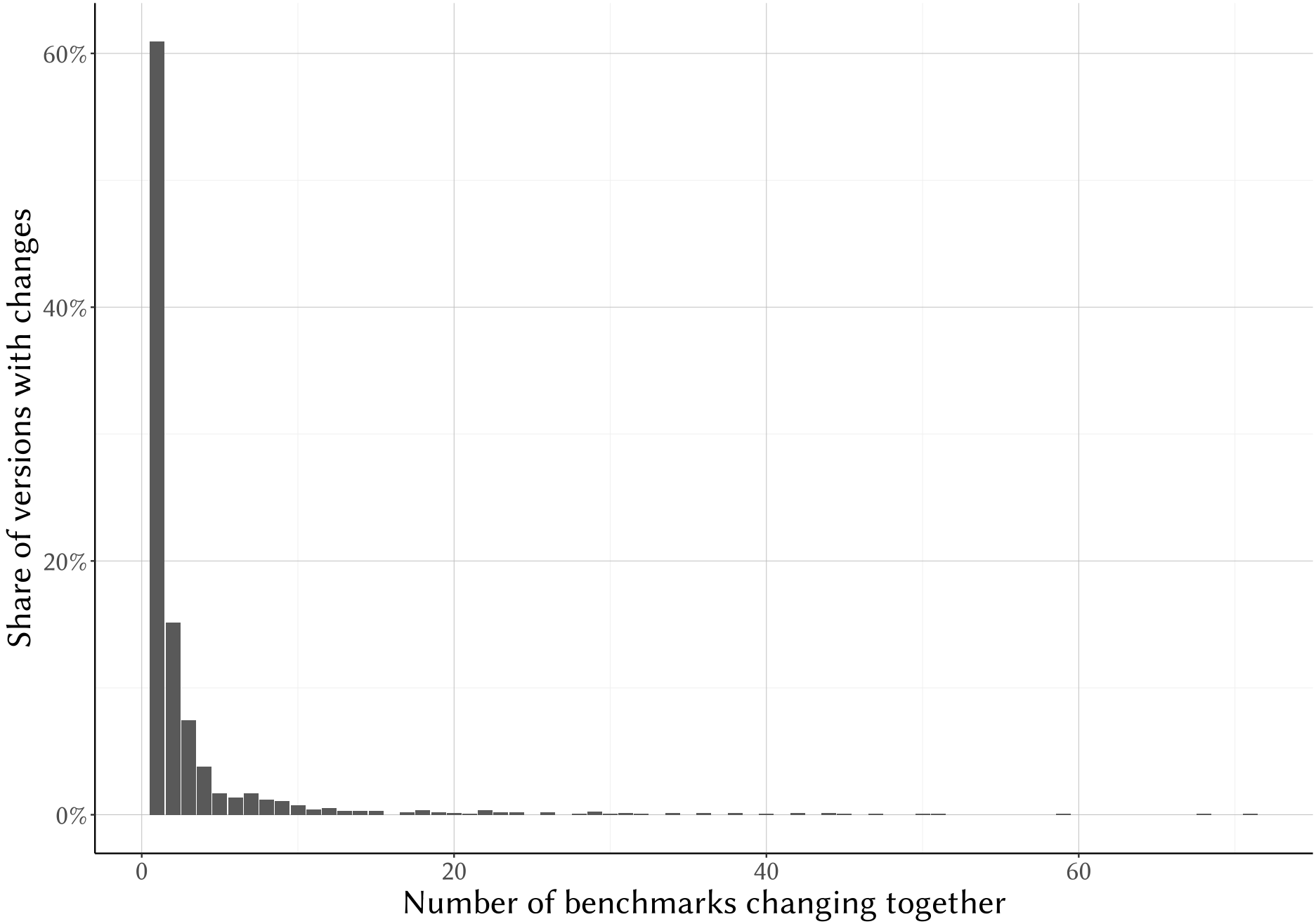
Input All detected performance changes across measurement history.

Computation Count how many benchmarks are impacted by particular compiler version commits.

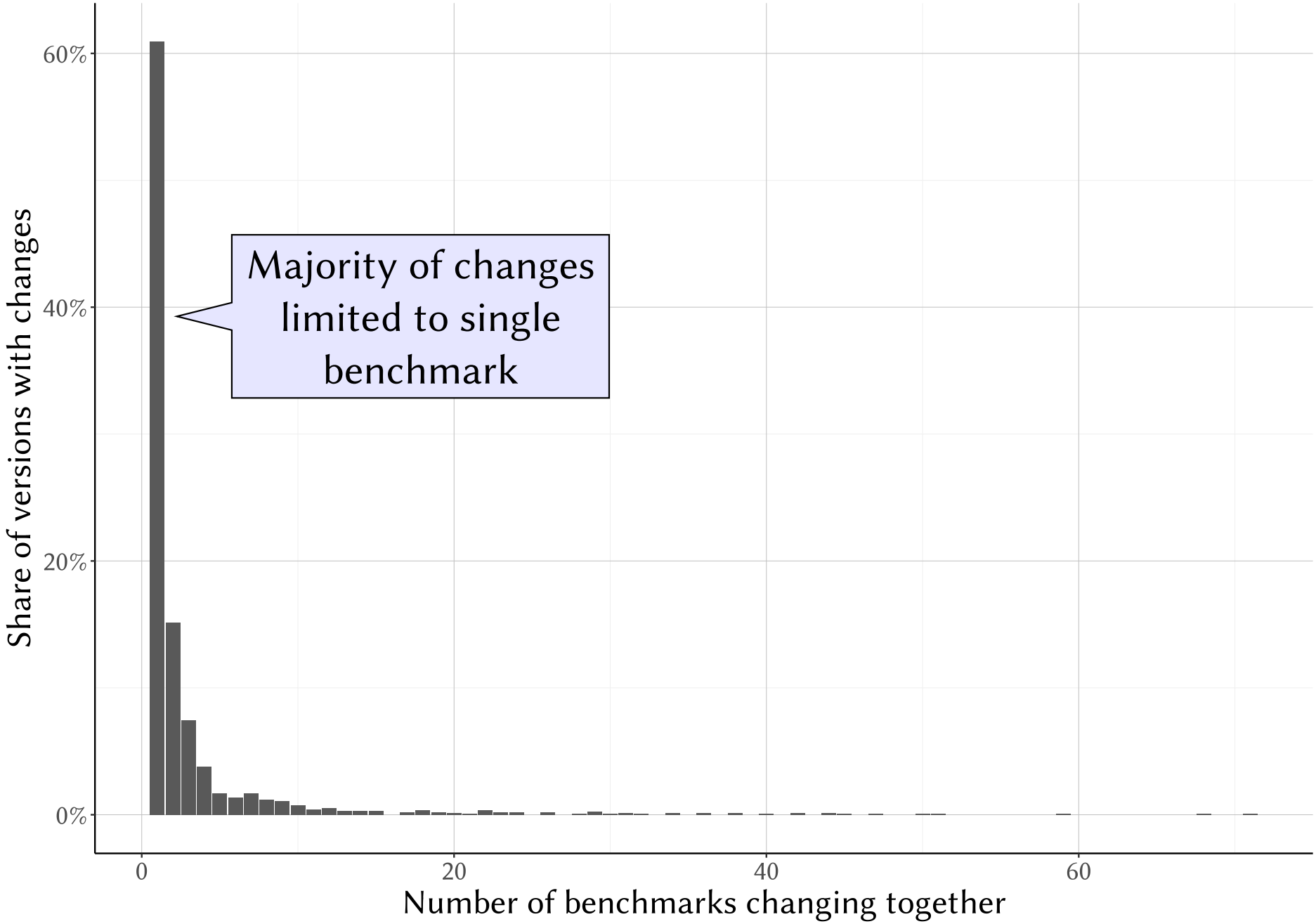
X axis Count of benchmarks changing together on the same commit.

Y axis Share of compiler versions with that count.

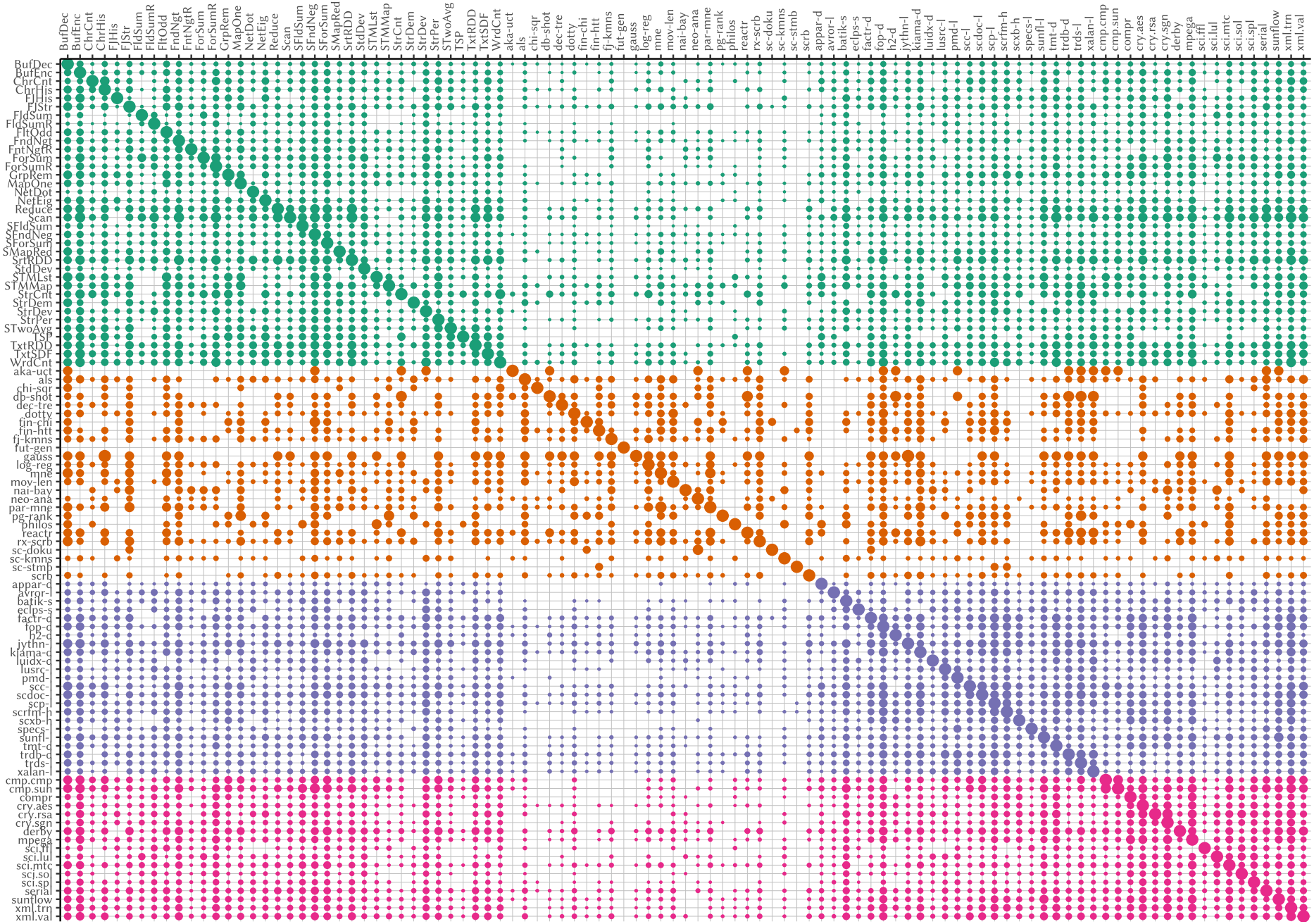
Do We Have Too Many Benchmarks ?



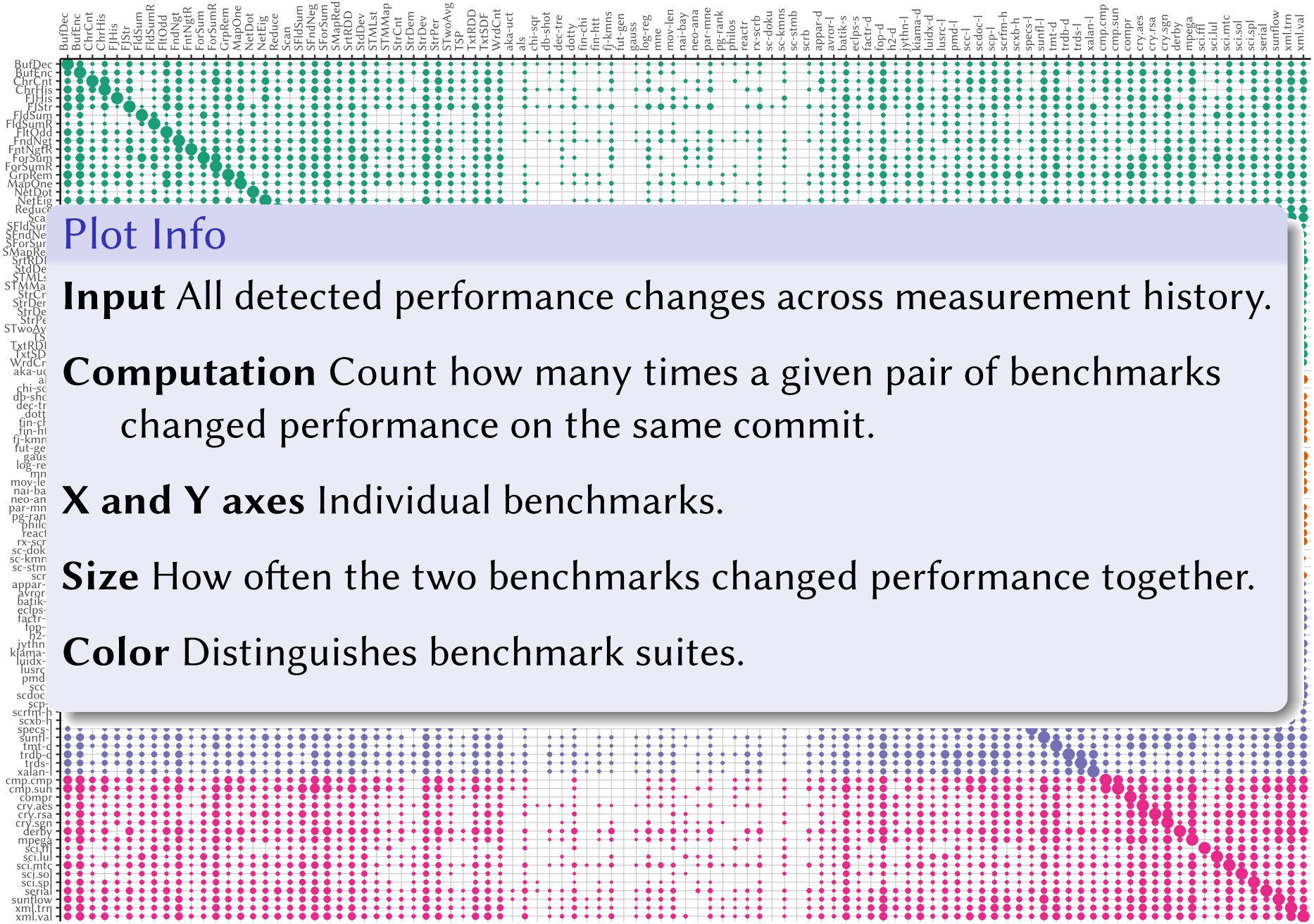
Do We Have Too Many Benchmarks ?



Do Benchmarks Change Together ?



Do Benchmarks Change Together ?



Plot Info

Input All detected performance changes across measurement history.

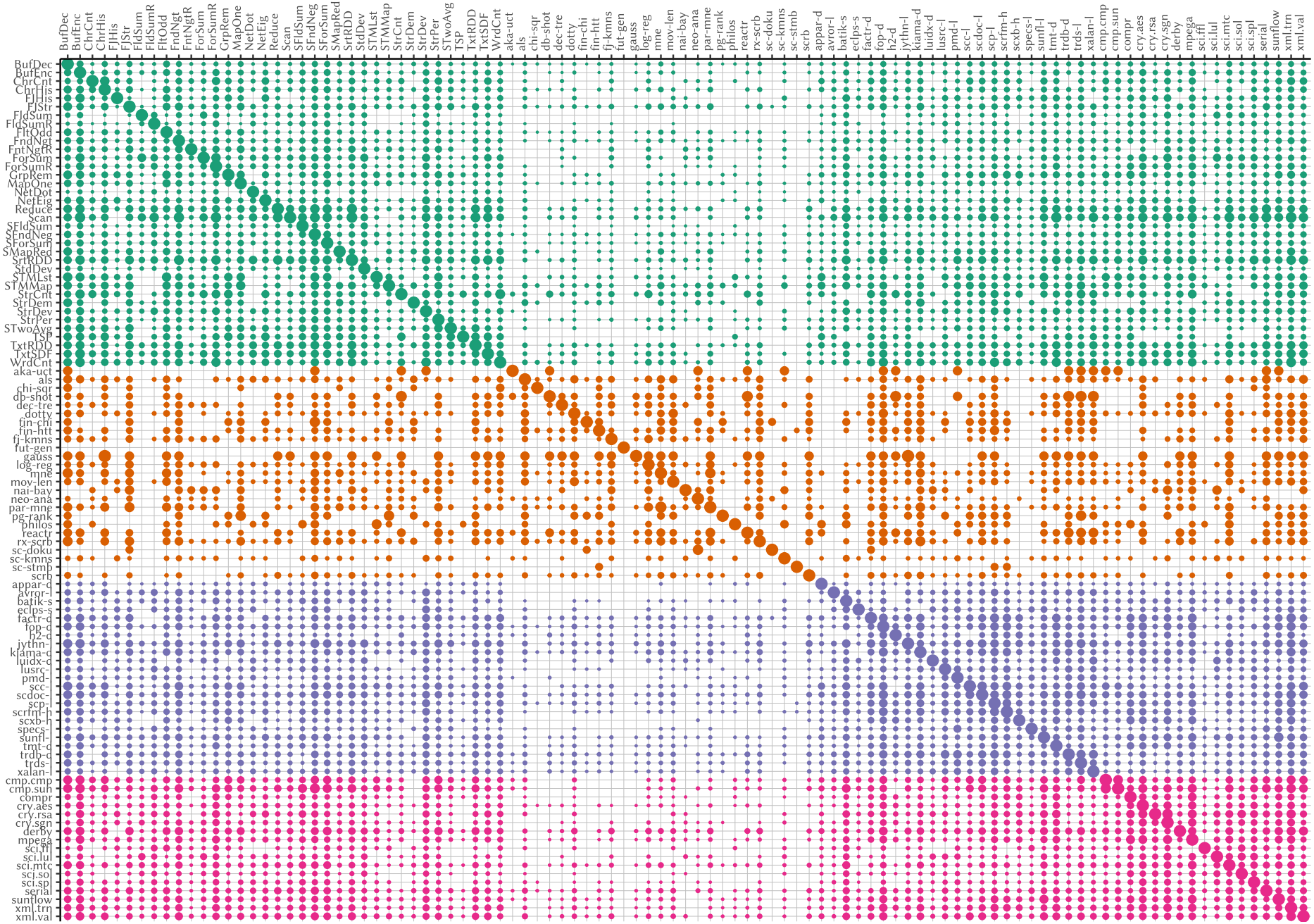
Computation Count how many times a given pair of benchmarks changed performance on the same commit.

X and Y axes Individual benchmarks.

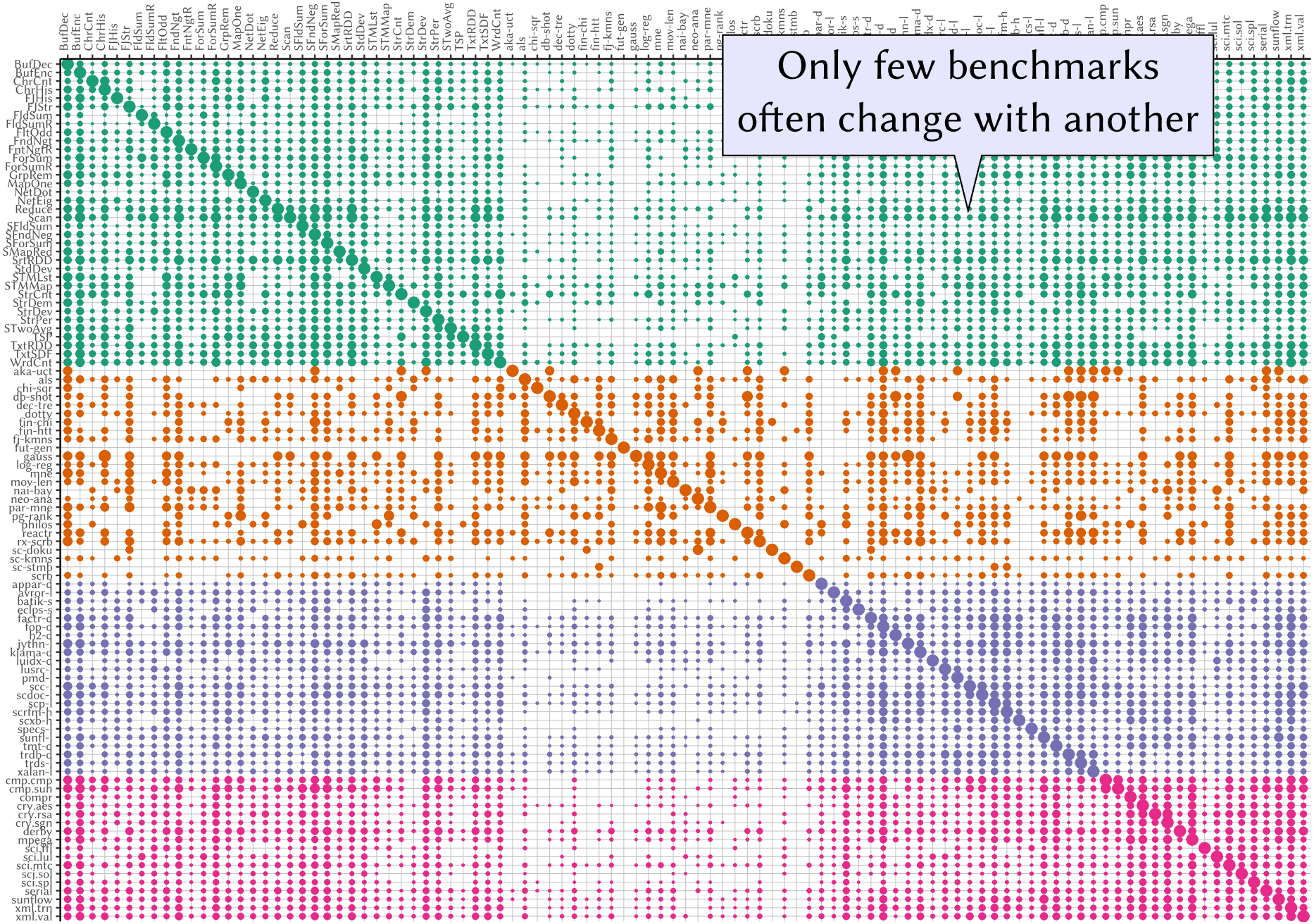
Size How often the two benchmarks changed performance together.

Color Distinguishes benchmark suites.

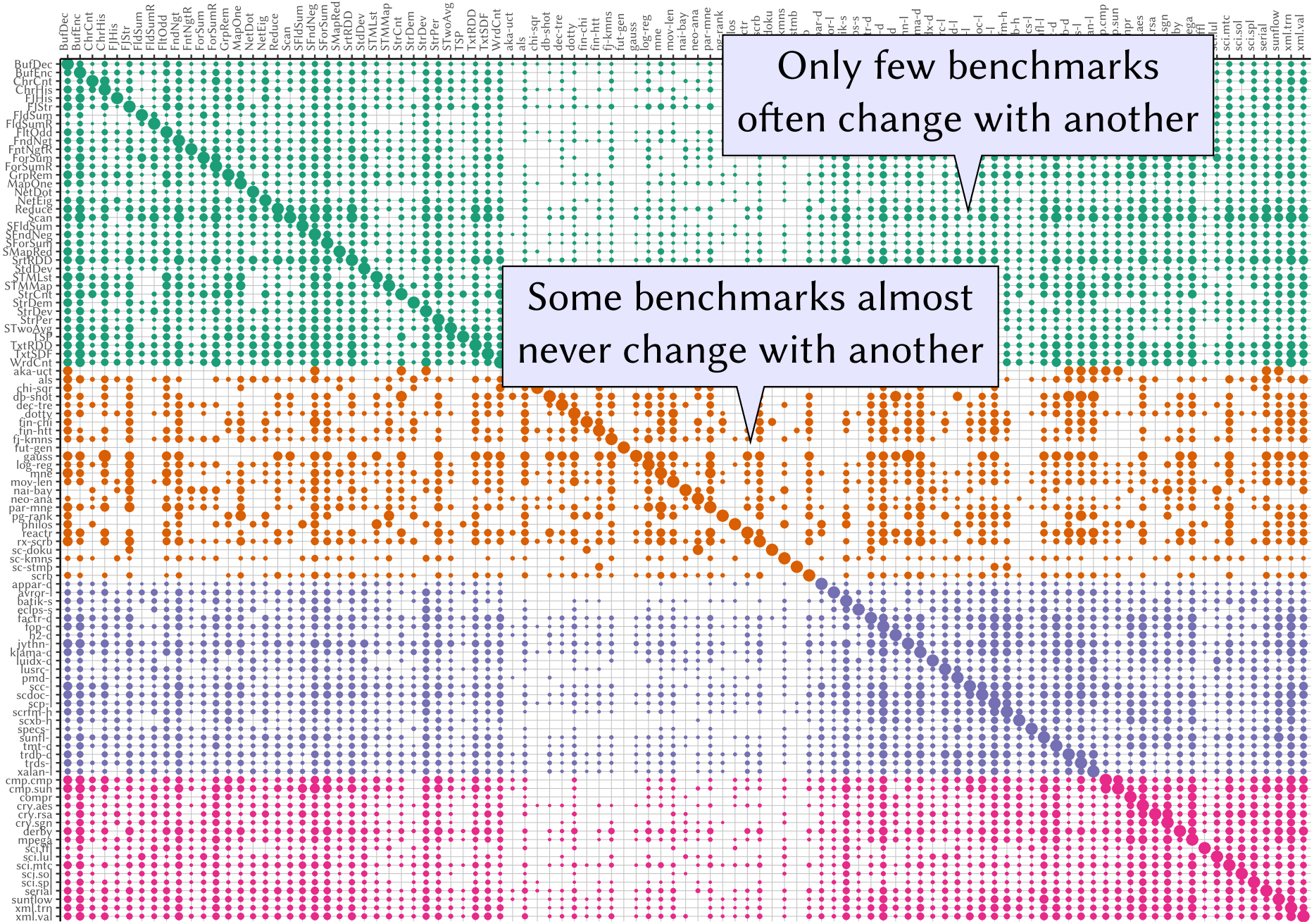
Do Benchmarks Change Together ?



Do Benchmarks Change Together ?



Do Benchmarks Change Together ?



Do Benchmarks Change Together ?