Tracking Performance of Graal on Public Benchmarks

Lubomír Bulej Vojtěch Horký Petr Tůma

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University

2018 – 2021

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

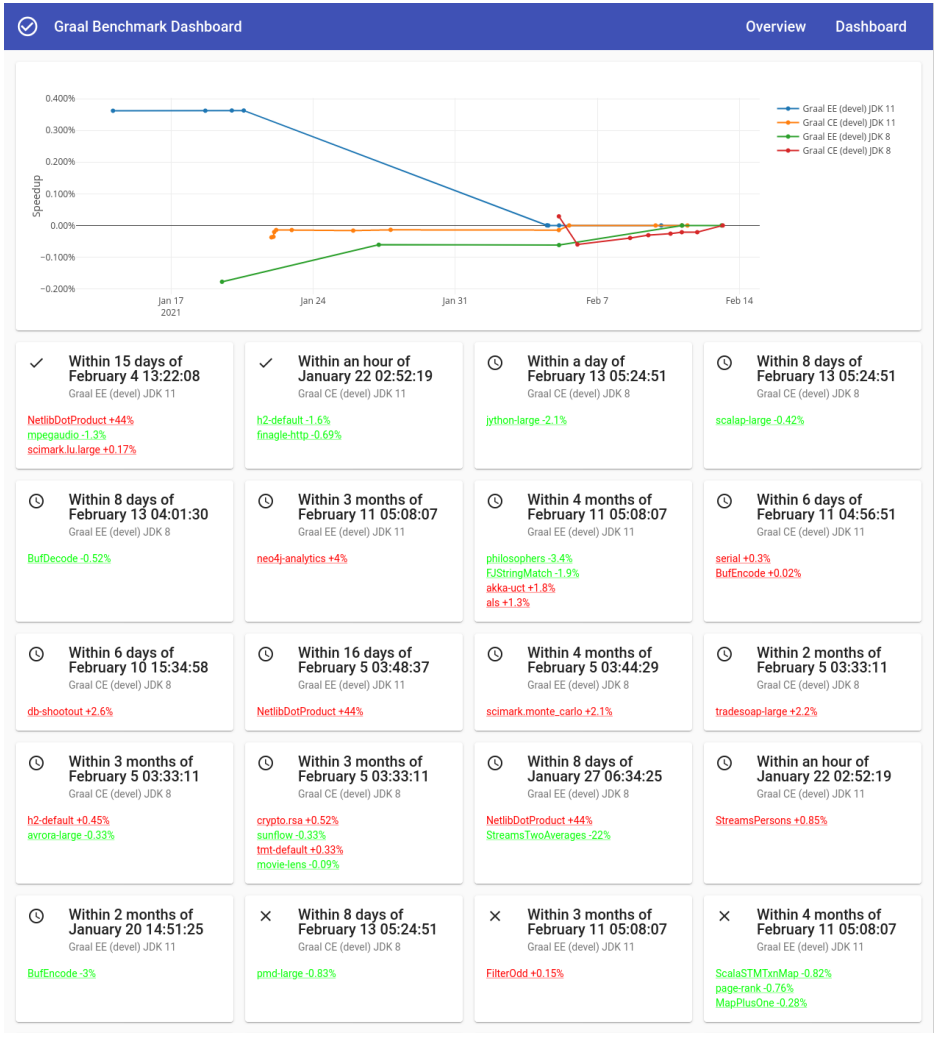
The data may be influenced by mistakes we are not aware of.

Performance Testing Goal ?

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

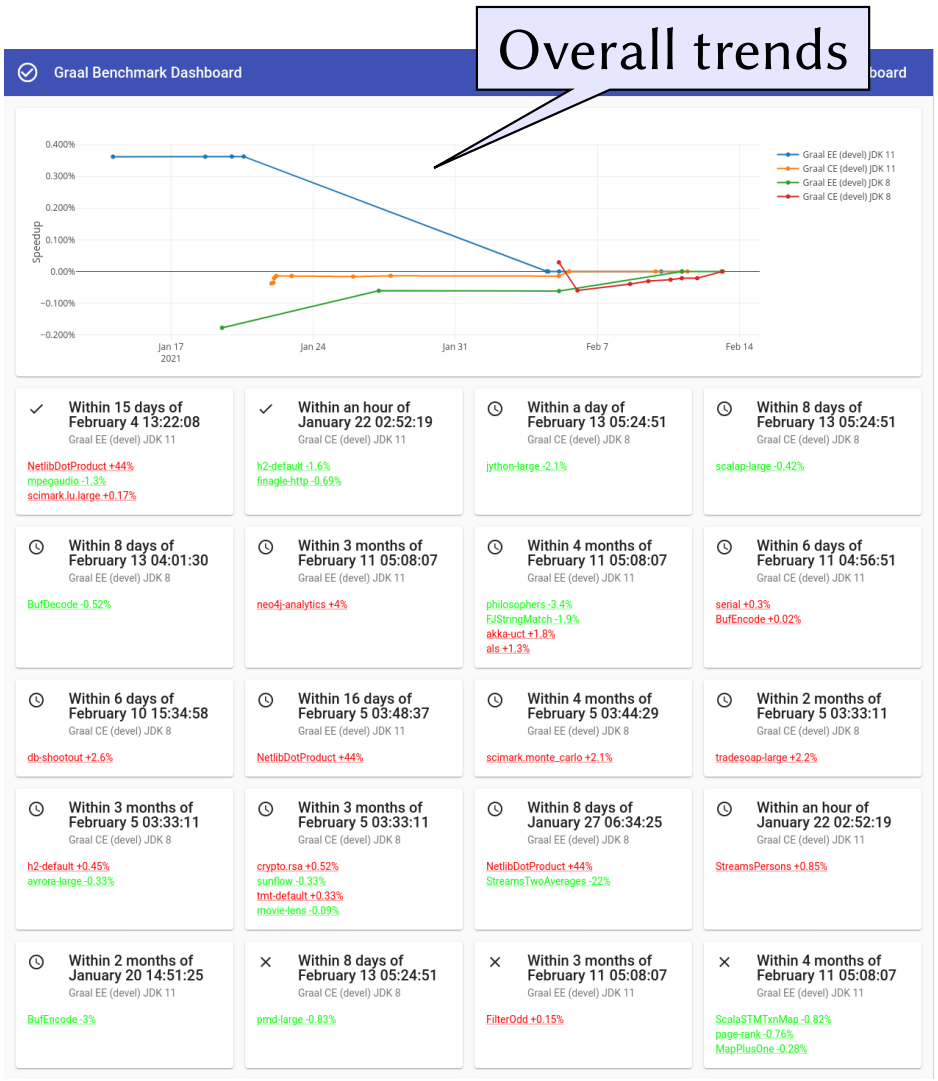
Performance Testing Goal ?

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



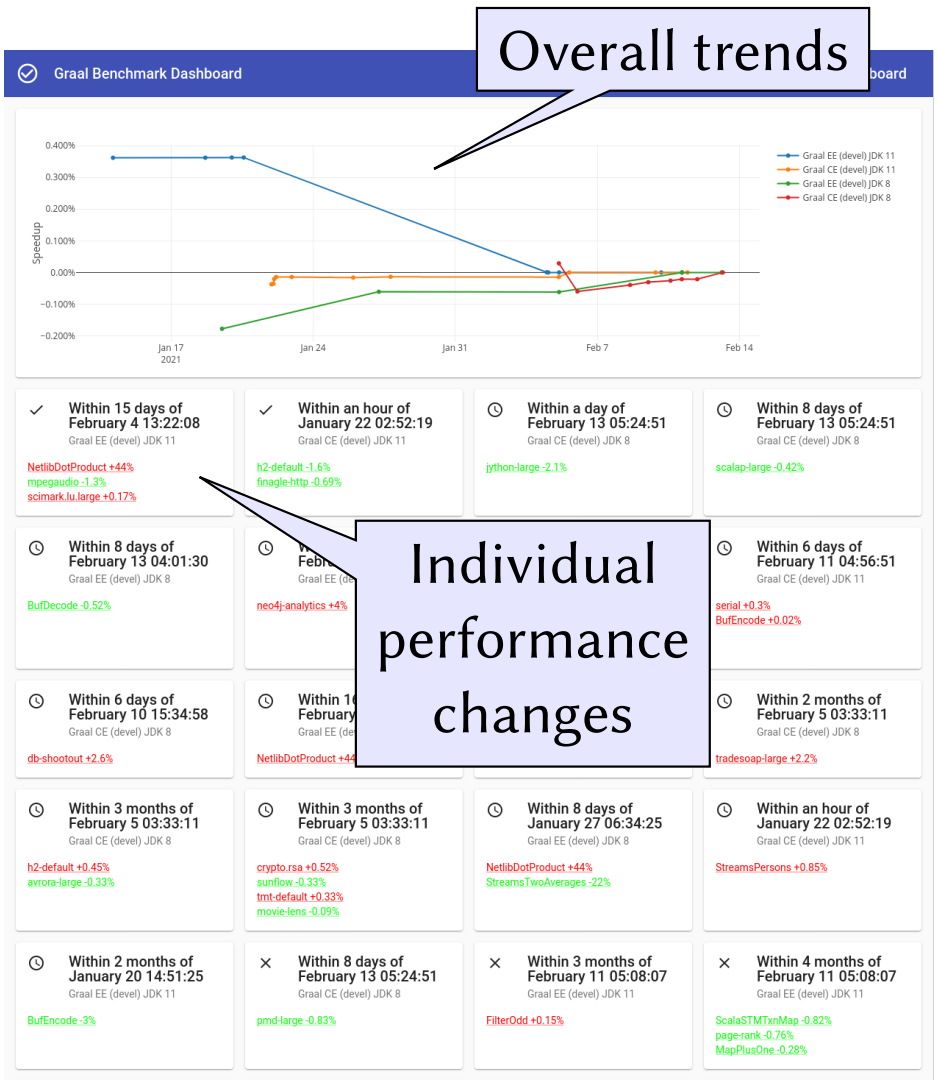
Performance Testing Goal ?

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



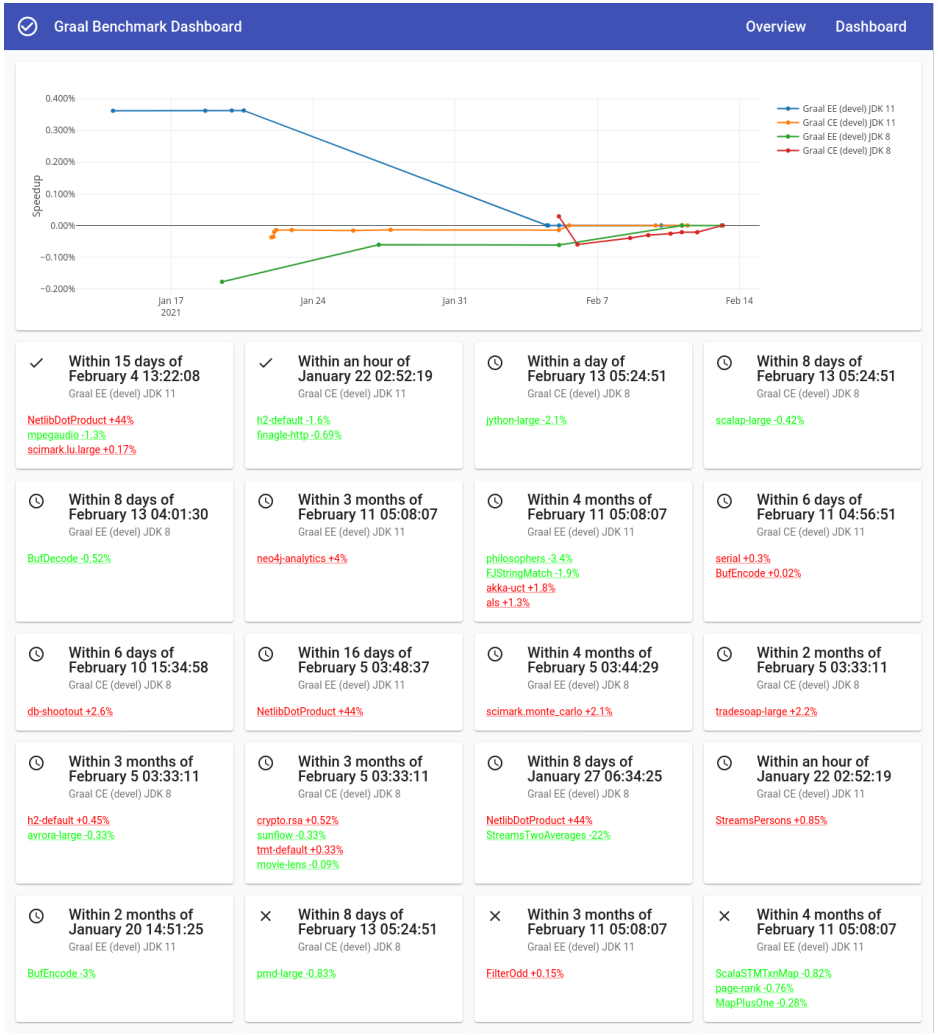
Performance Testing Goal ?

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



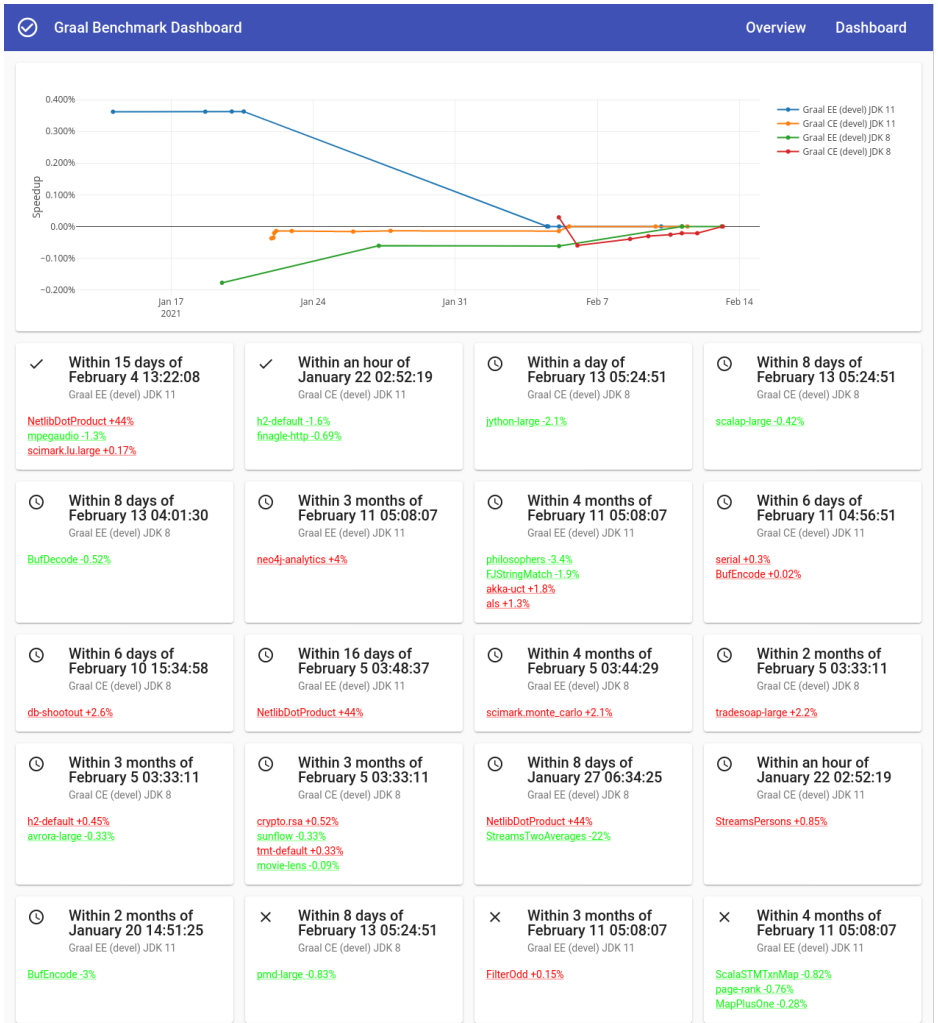
Performance Testing Goal ?

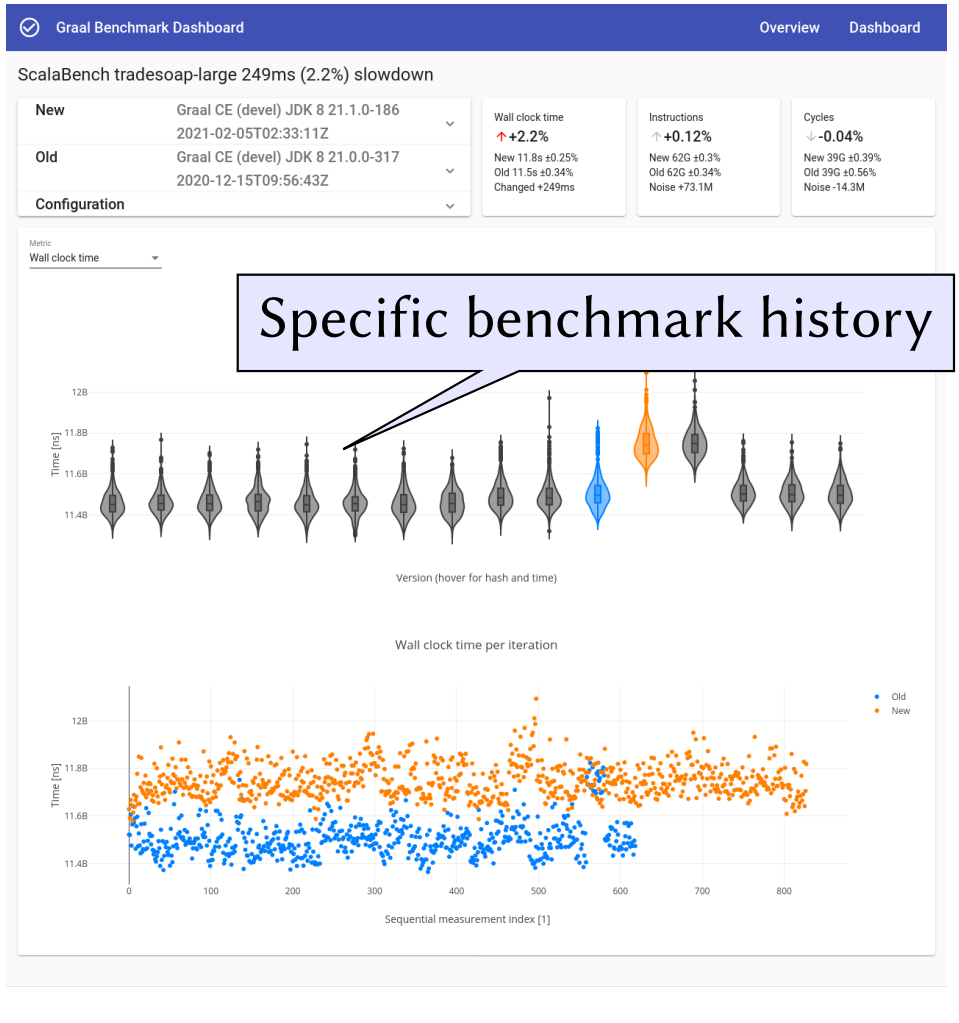
Make performance testing roughly the same as standard (functional) regression testing.



Performance Testing Goal ?

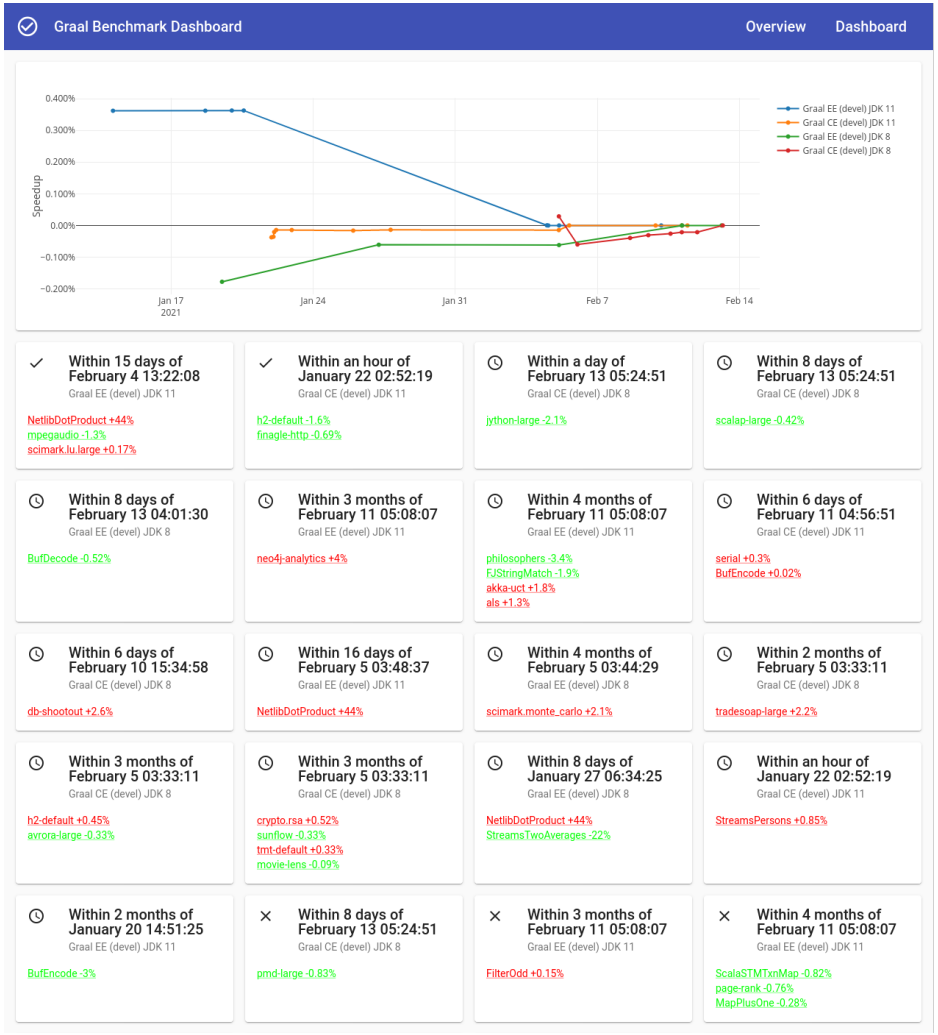
Make performance testing roughly the same as standard (functional) regression testing.

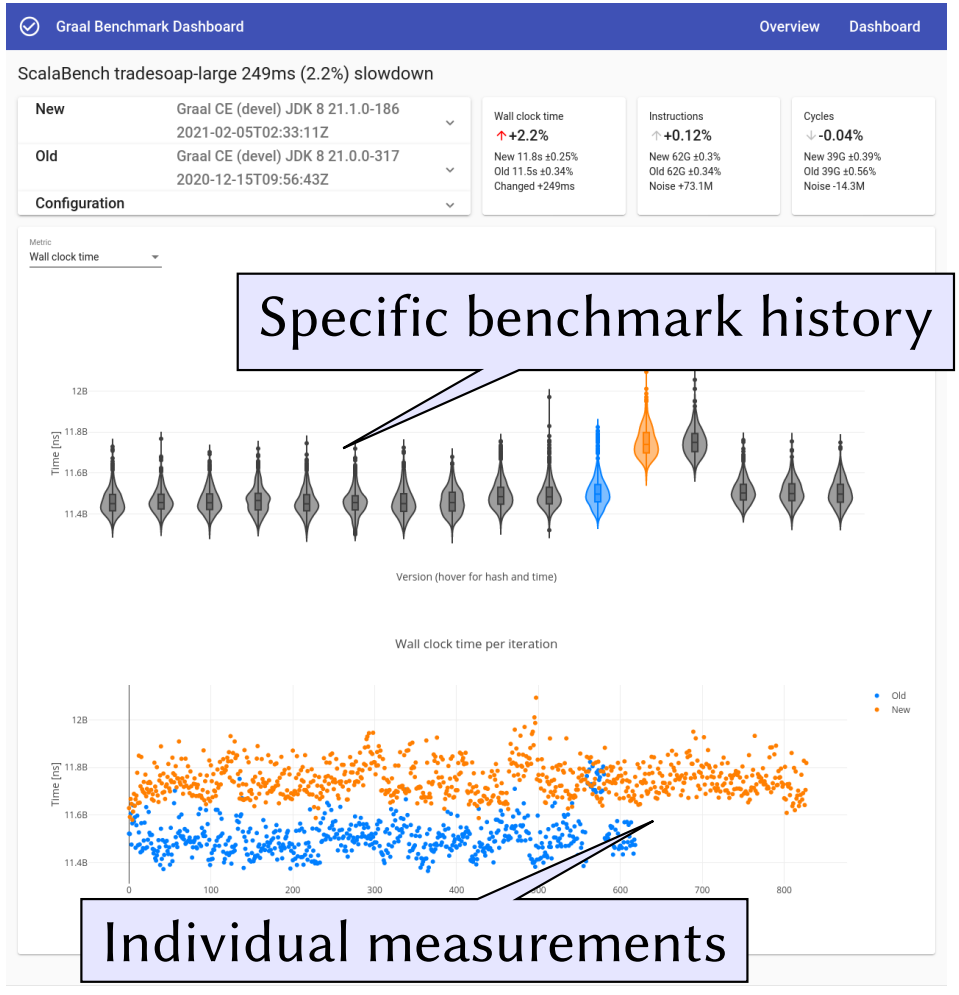




Performance Testing Goal ?

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





Dashboard Internals I

Platforms

- GraalVM CE and EE with OpenJDK and HotSpot JDK 8 and 11
- Only top level merge commits into master
- ... around 6000 versions last year

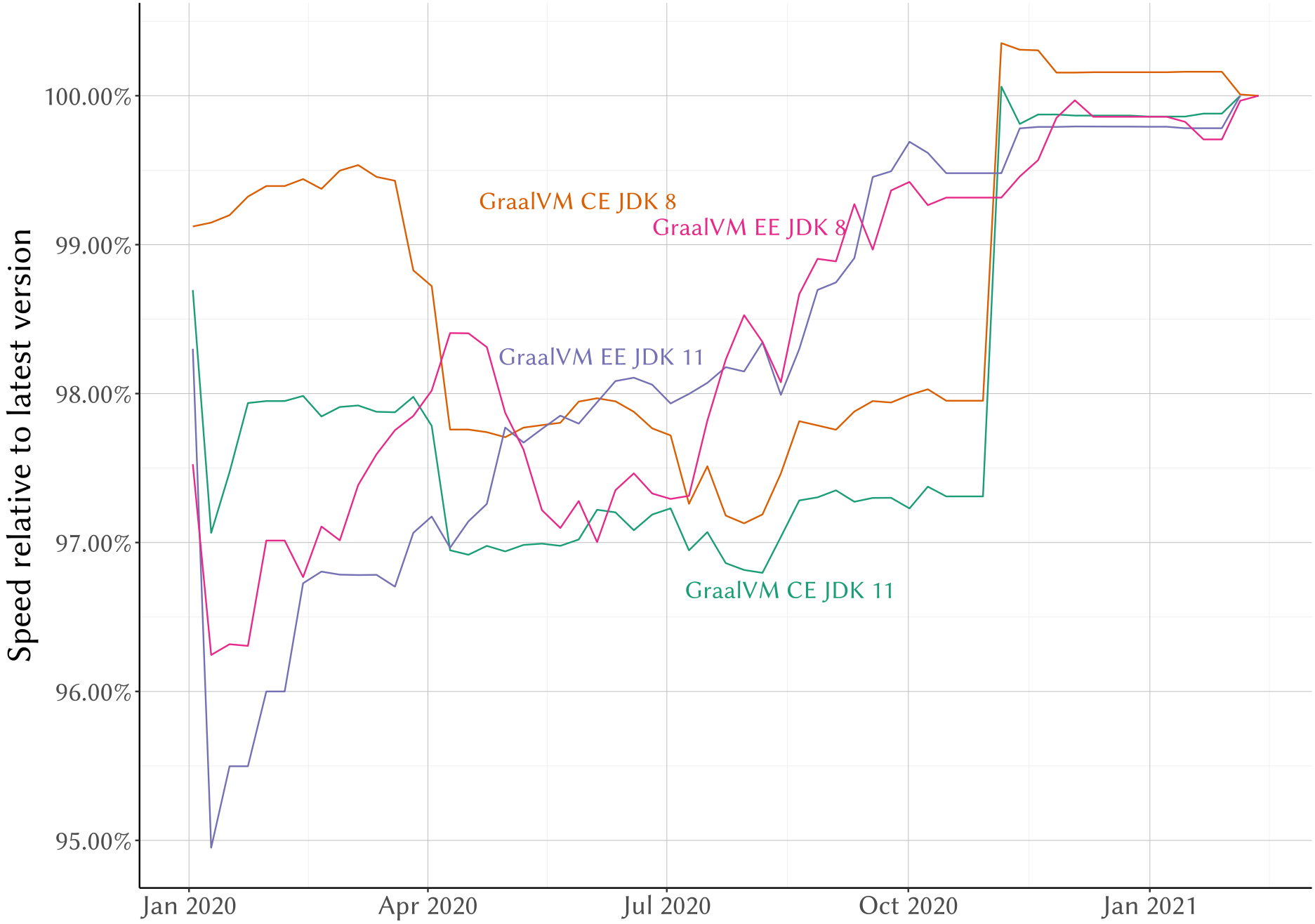
Benchmarks

- ScalaBench (includes DaCapo) `https://scalabench.org`
- SPECjvm2008 (modified) `https://spec.org/jvm2008`
- Renaissance 0.10 `https://renaissance.dev`
- Plus internal microbenchmarks
- ... around 130 workloads in all

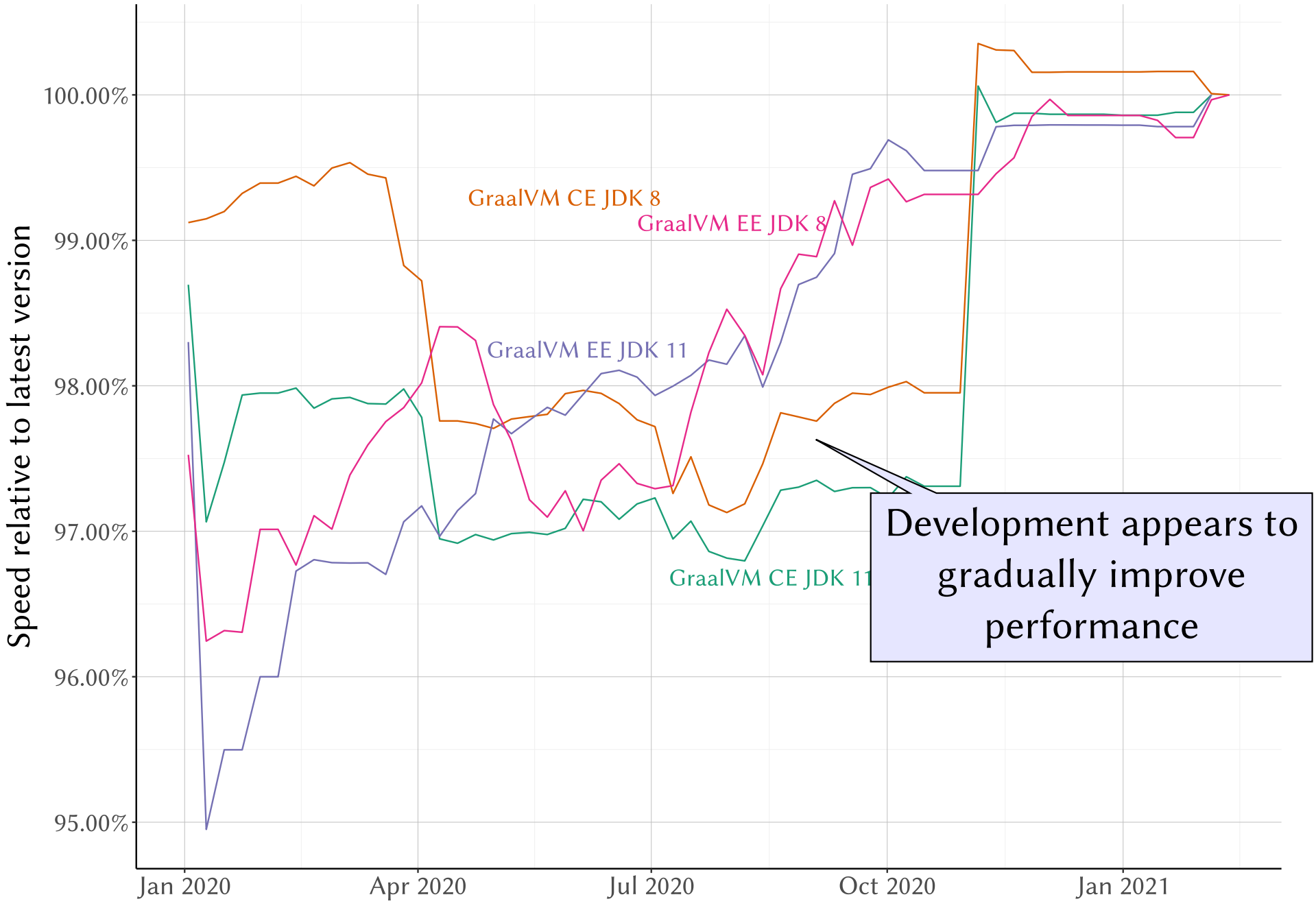
Hardware

- ... around 40 dedicated servers

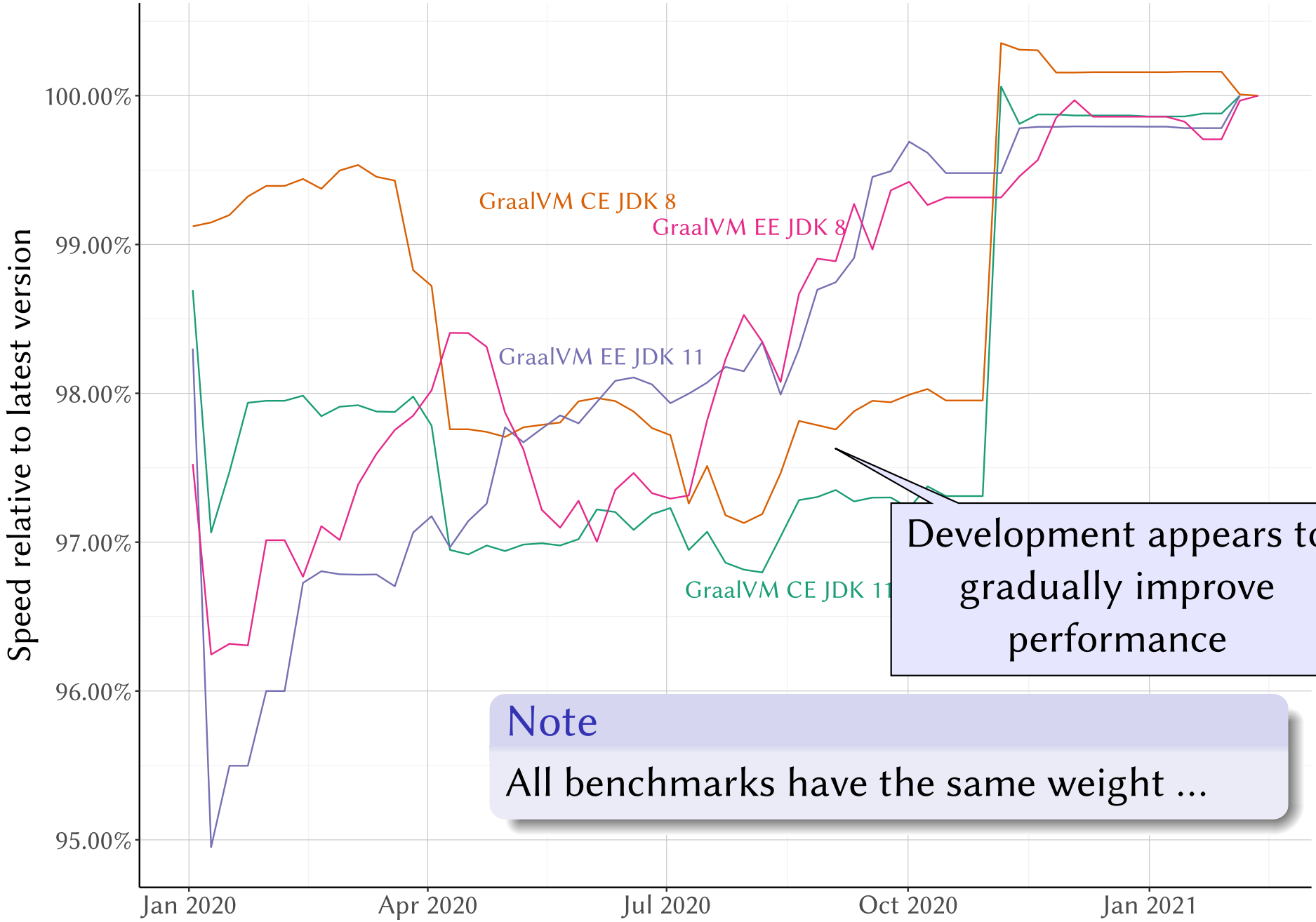
Summary Performance History



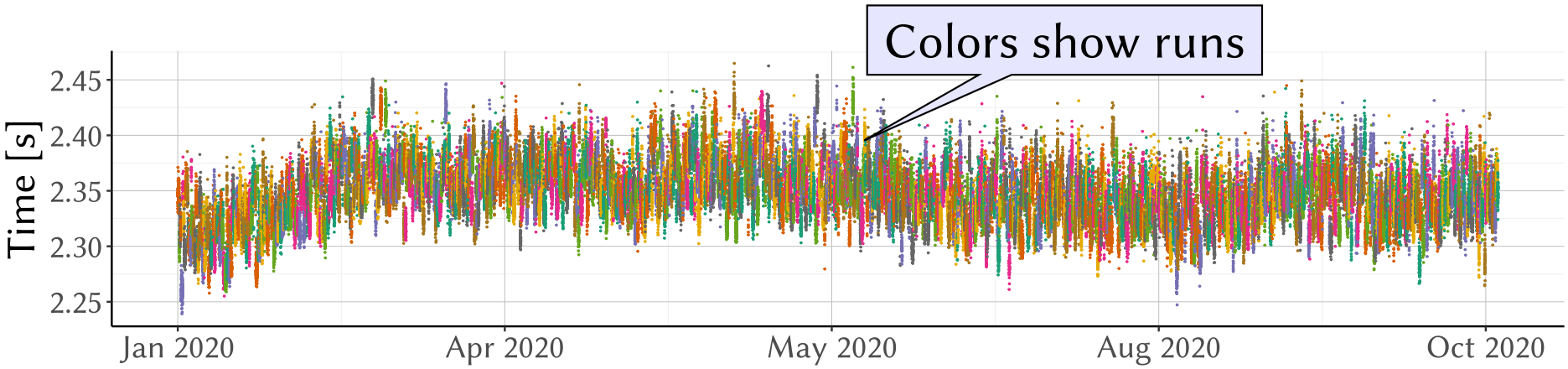
Summary Performance History



Summary Performance History

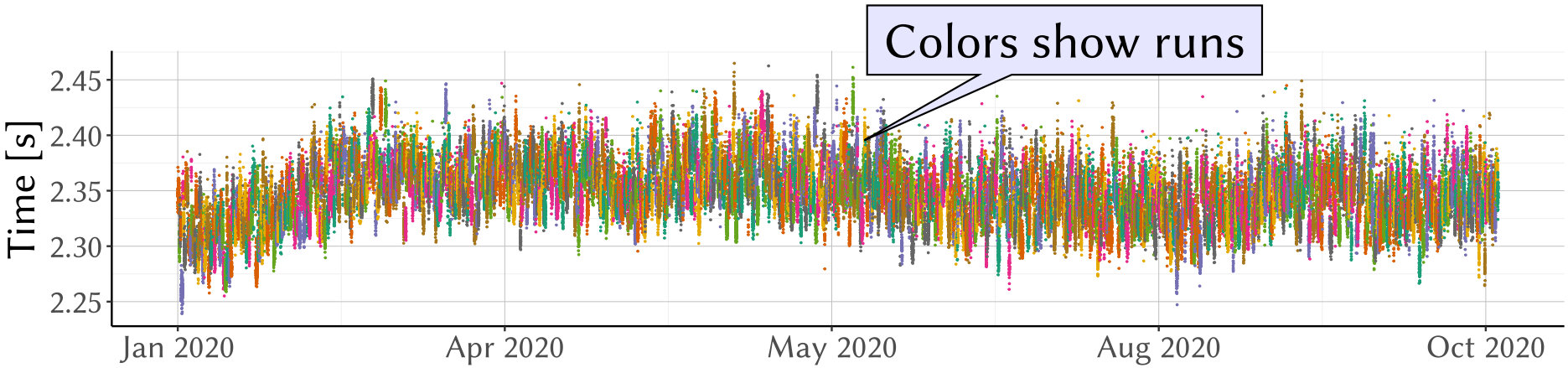


Detecting Changes



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

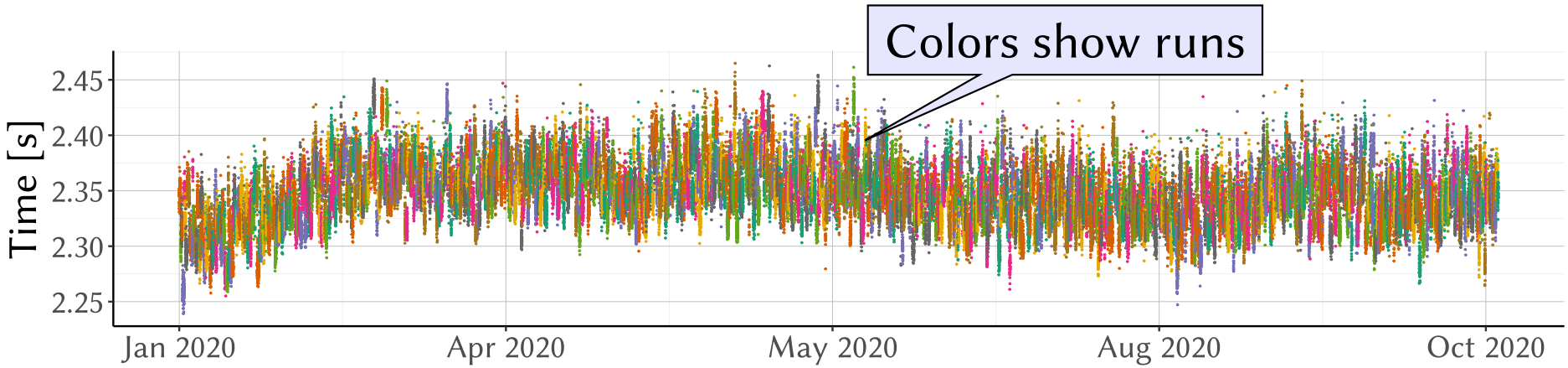
Detecting Changes



A time series change point detection problem with a few twists

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

Detecting Changes

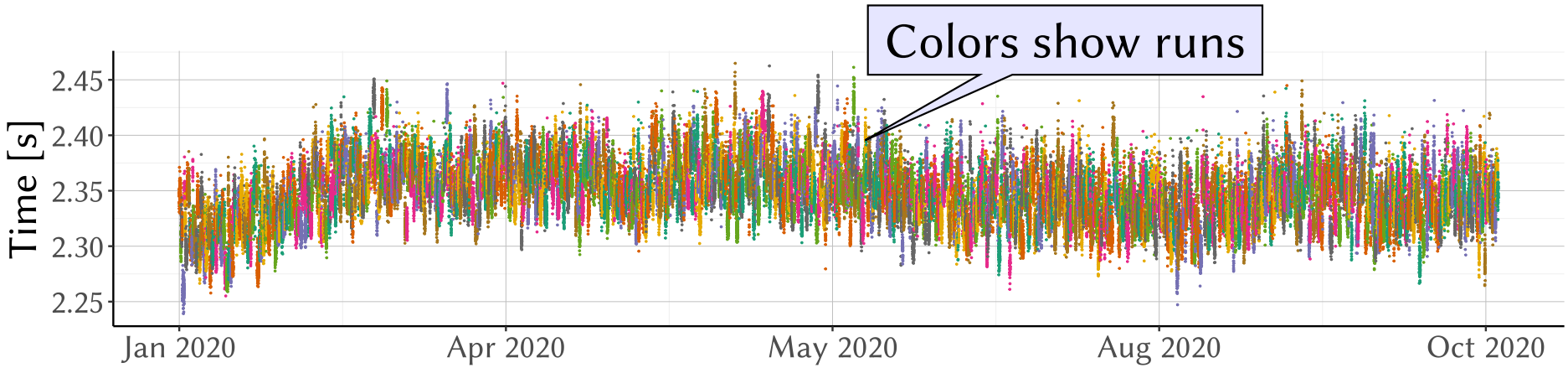


A time series change point detection problem with a few twists

- We have more correlated time series rather than just one

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

Detecting Changes

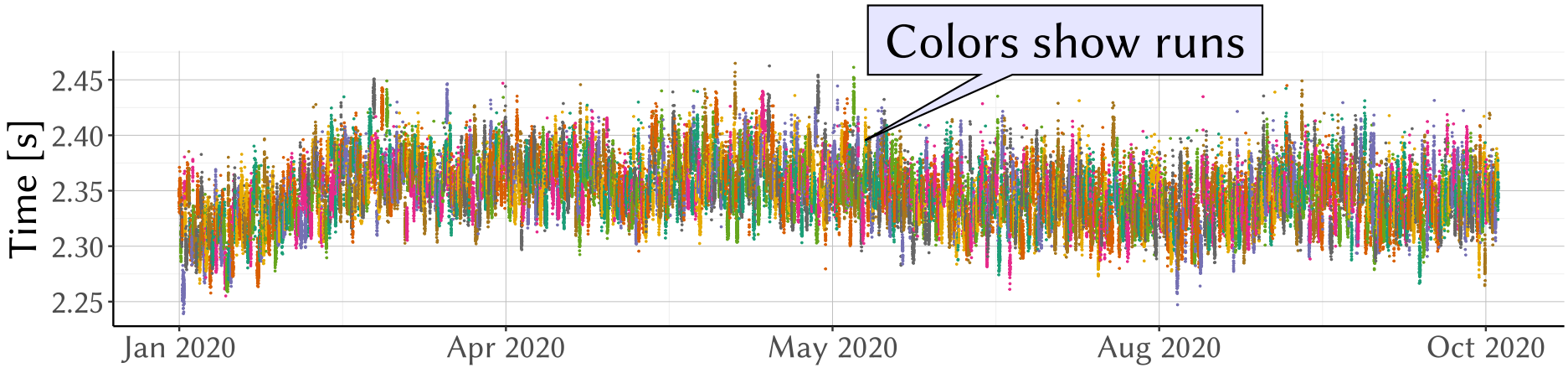


A time series change point detection problem with a few twists

- We have more correlated time series rather than just one
- We can add more data points to any version if required

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

Detecting Changes

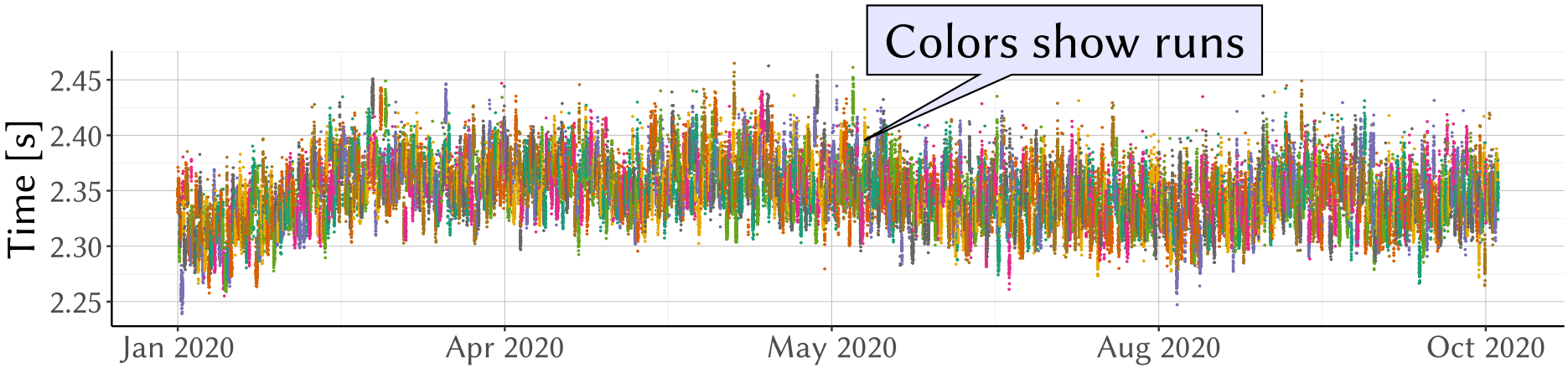


A time series change point detection problem with a few twists

- 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

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

Detecting Changes

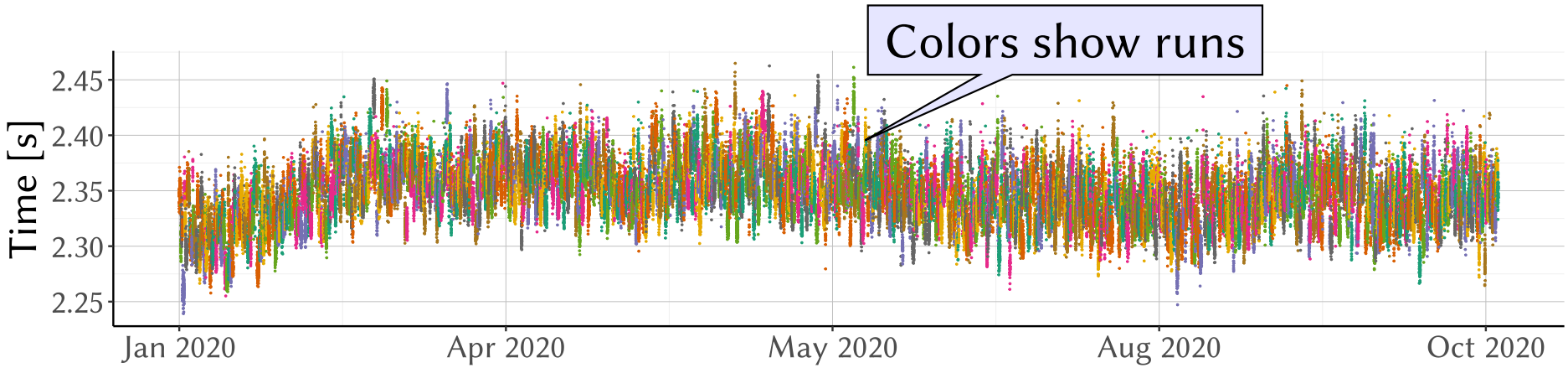


A time series change point detection problem with a few twists

- 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

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

Detecting Changes



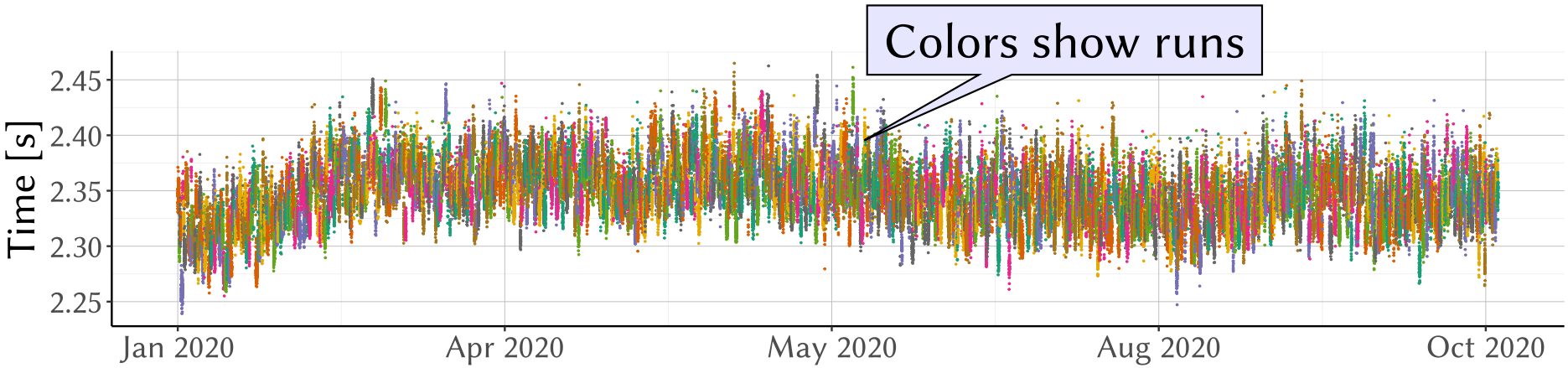
A time series change point detection problem with a few twists

- 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 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]	[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]	2%	89%	11%
[illegible]	1%	100%	0%
[illegible]	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 ?

Detection mostly reliable enough

Most benchmarks exhibit changes

Renaissance 0.10

bench	R	D	I
aka-uct	1%	100%	0%
als	5%	100%	0%
chi-sqr	1%	100%	0%
dec-tree	1%	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	1%	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%
[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%

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%	
SMapRed	3%	43%	
STwoAvg	4%	60%	
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%
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%

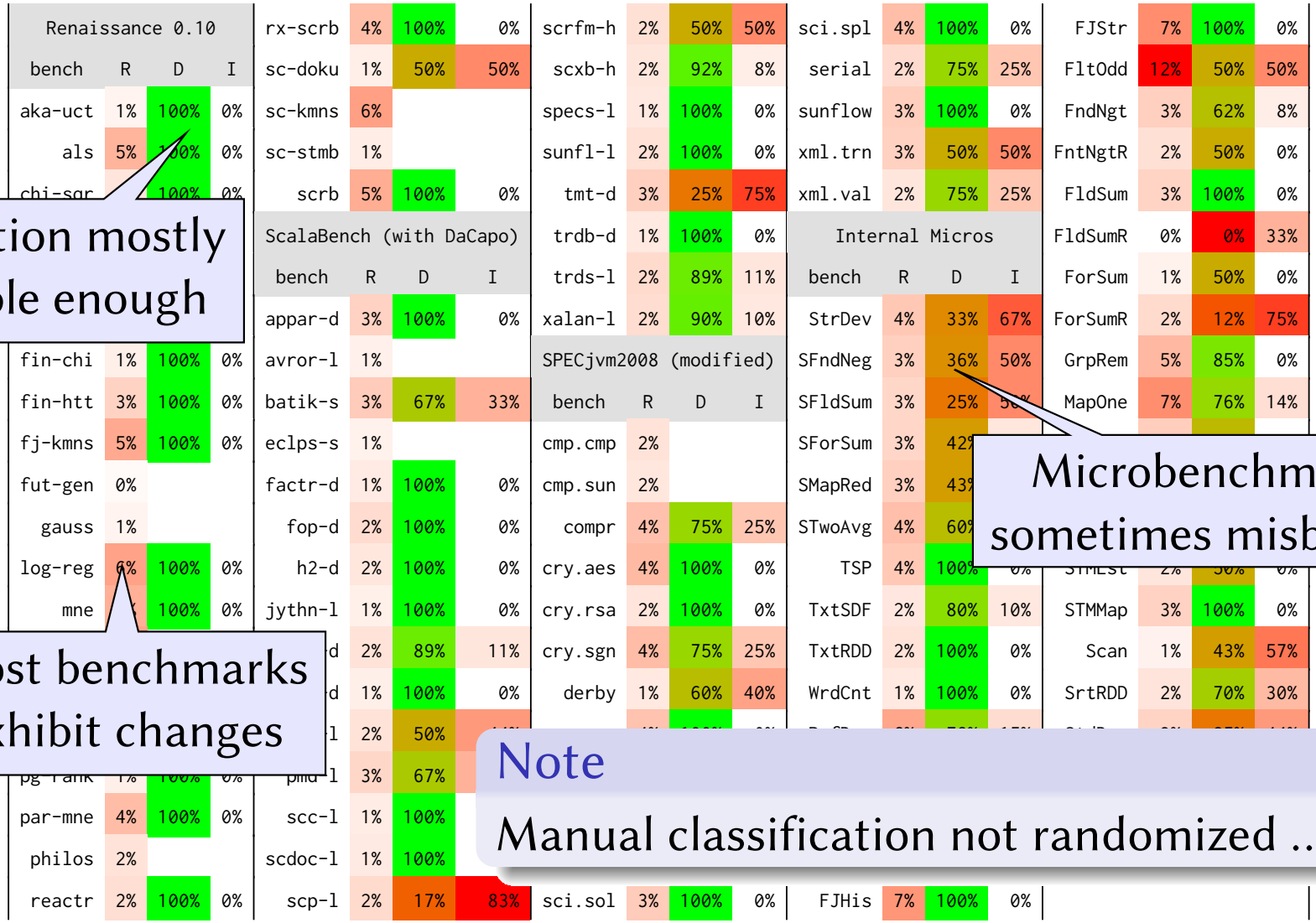
R - versions with changes

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%
	2%	89%	11%
	1%	100%	0%
	2%	50%	
pmd-l	3%	67%	
scc-l	1%	100%	
scdoc-l	1%	100%	
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%
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%	
SForSum	3%	42%	
SMapRed	3%	43%	
STwoAvg	4%	60%	
TSP	4%	100%	
TxtSDF	2%	80%	10%
TxtRDD	2%	100%	0%
WrdCnt	1%	100%	0%
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%
STMMap	3%	100%	0%
Scan	1%	43%	57%
SrtRDD	2%	70%	30%

Detection mostly reliable enough

Most benchmarks exhibit changes

Microbenchmarks sometimes misbehave

Note

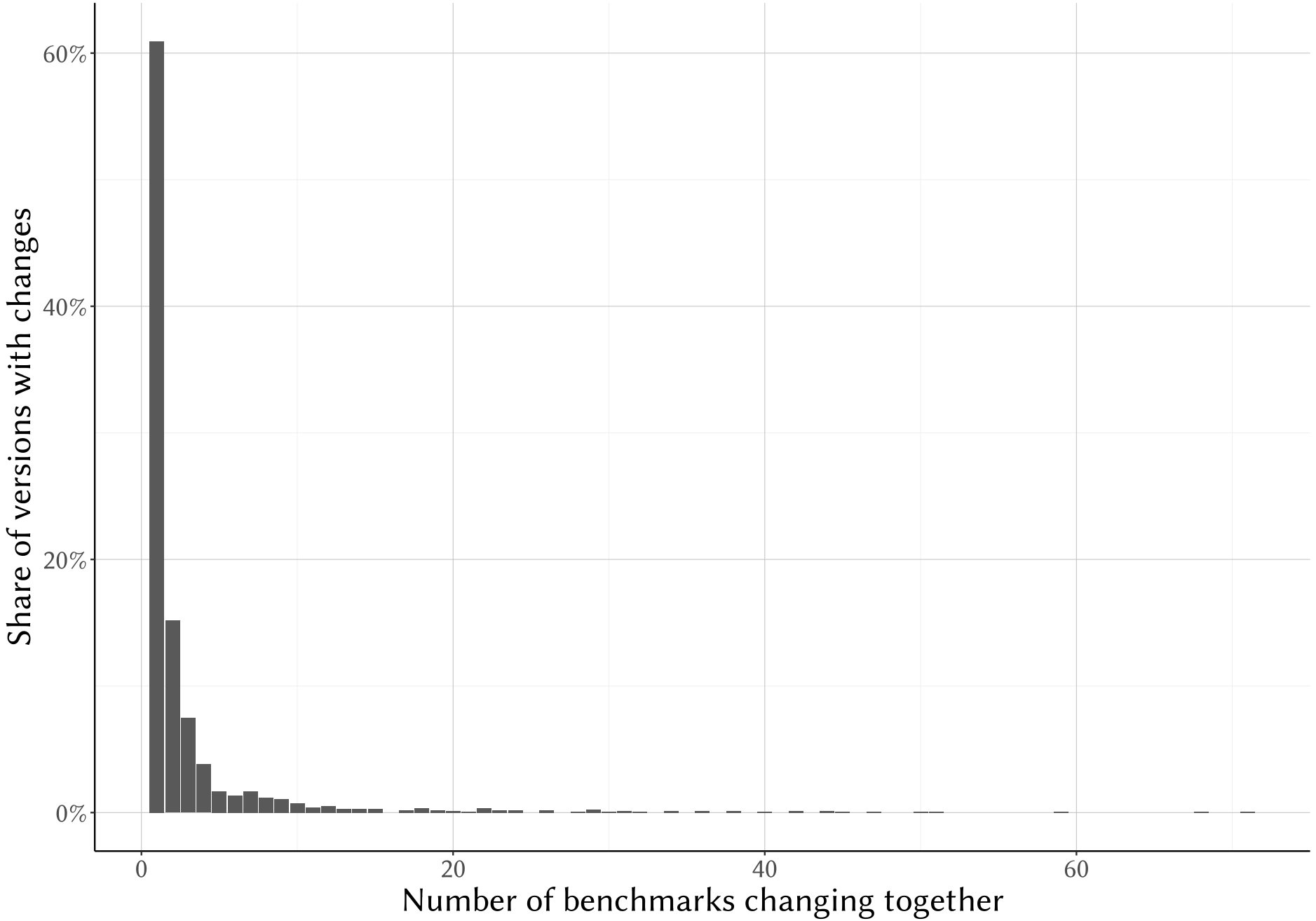
Manual classification not randomized ...

R - versions with changes

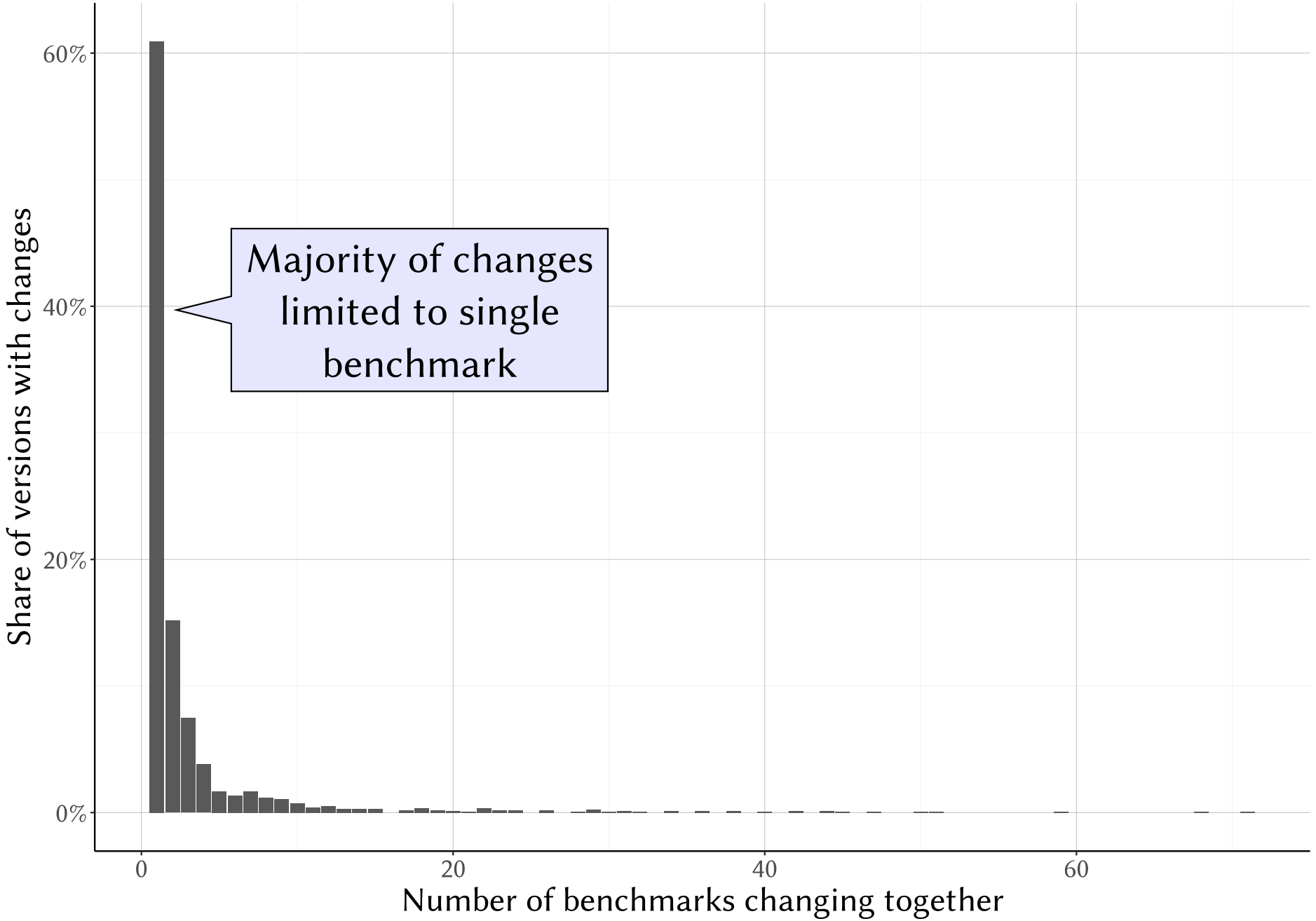
D - manually confirmed

I - invalid situations

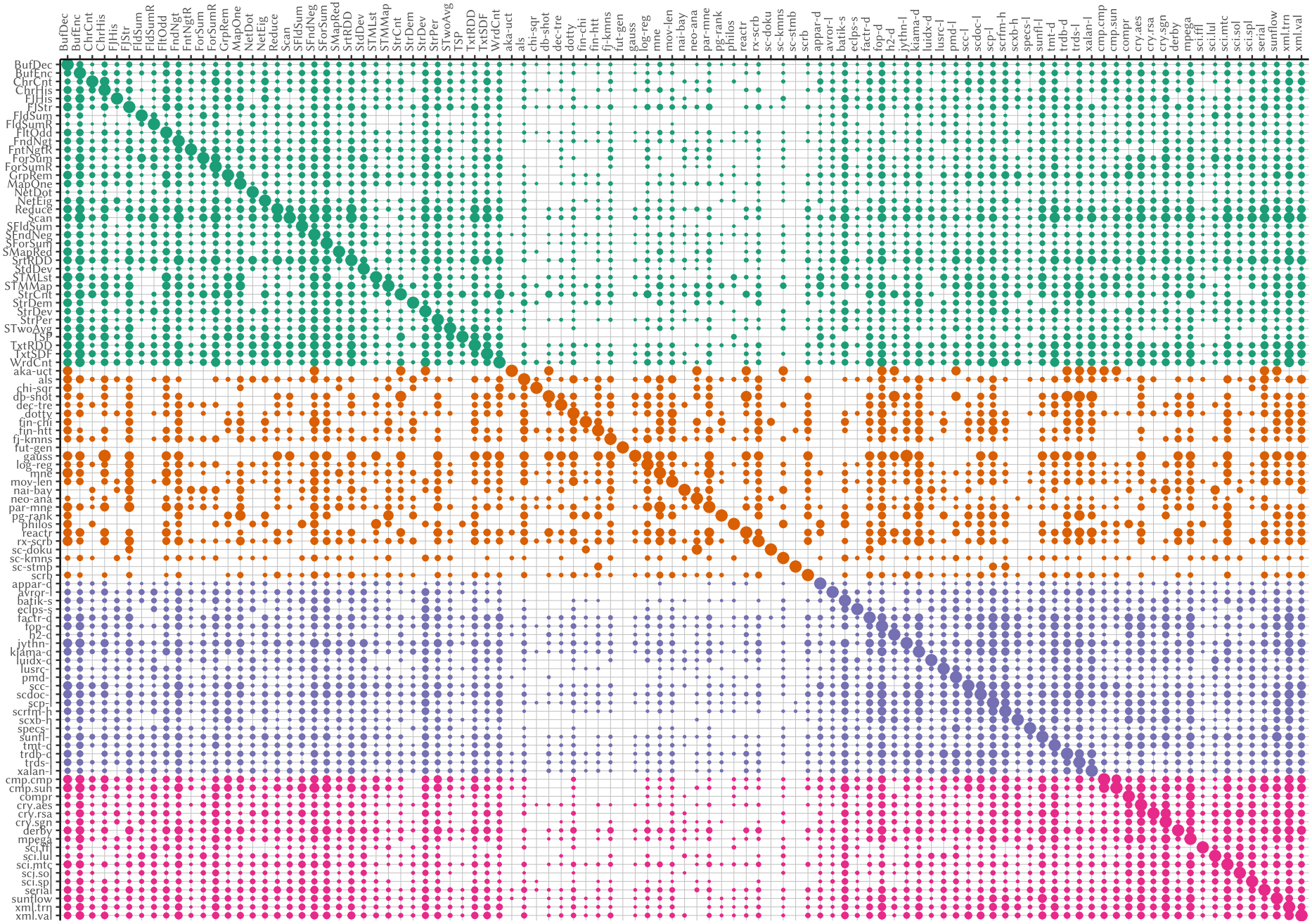
Do We Have Too Many Benchmarks ?



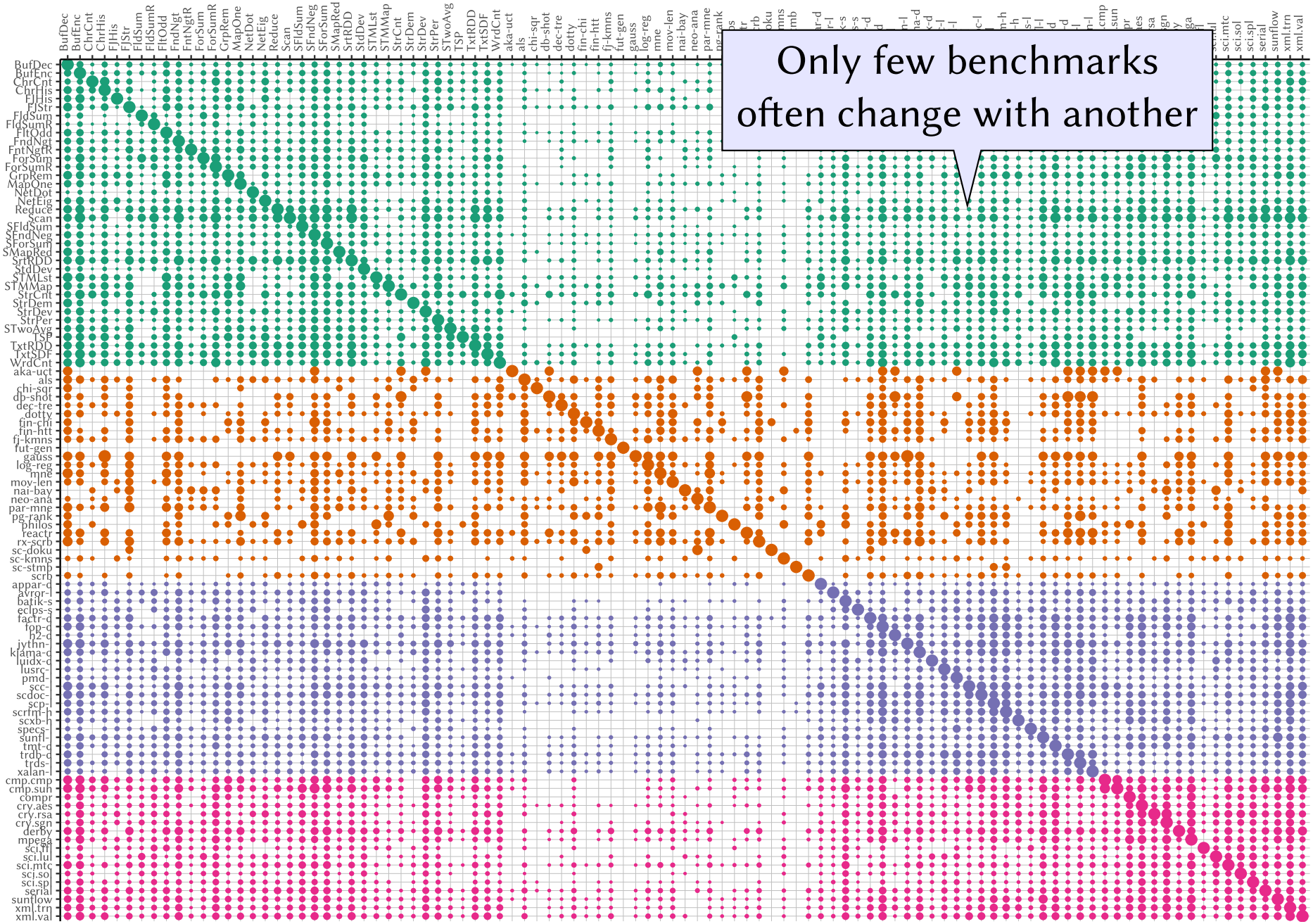
Do We Have Too Many Benchmarks ?



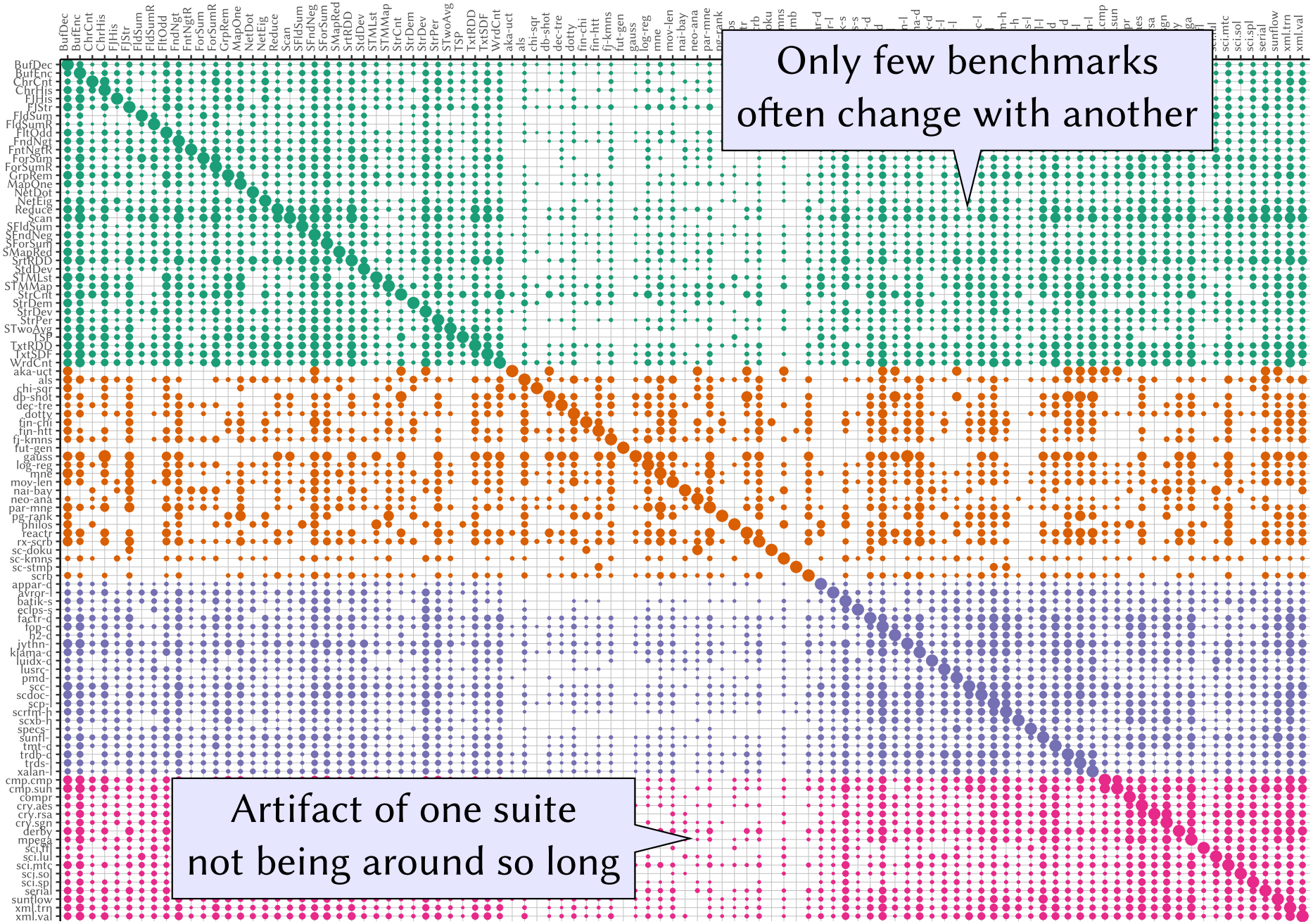
Do Benchmarks Change Together ?



Do Benchmarks Change Together ?



Do Benchmarks Change Together ?



Take Away So Far ...

We probably do not have too many (or even enough) benchmarks

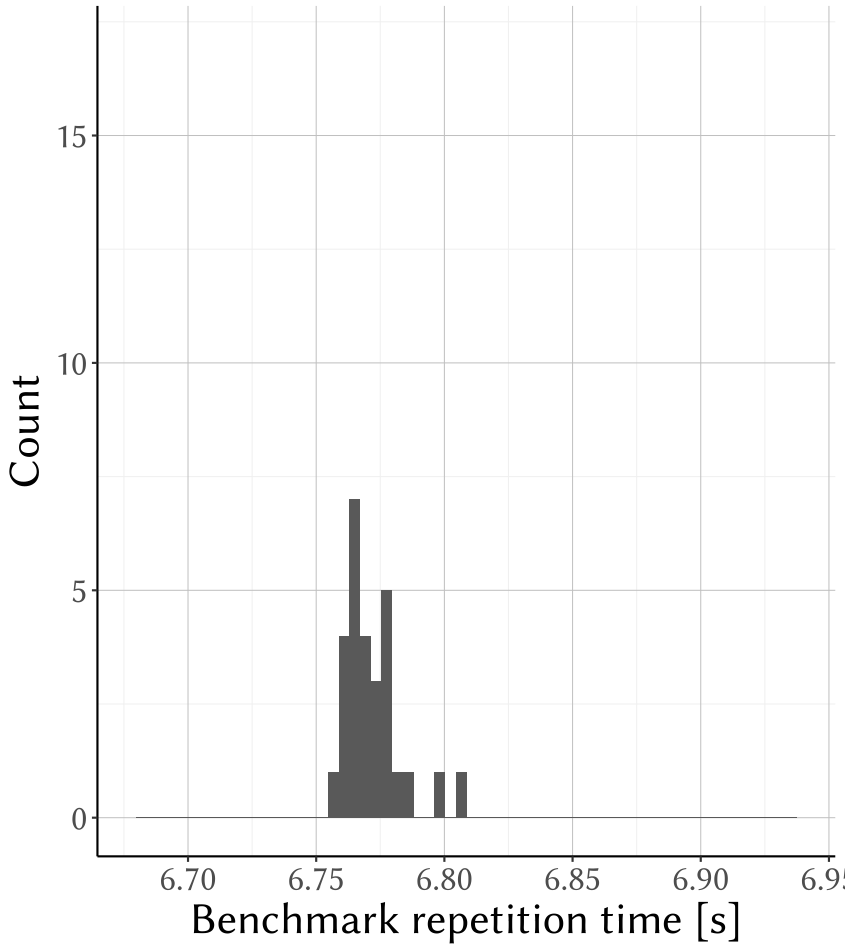
- Overlap in performance changes relatively rare
- Not really clear how to define coverage !

Change detection reliability per se not an issue

- But requires reasonable measurement procedure
- And some benchmarks may require special attention

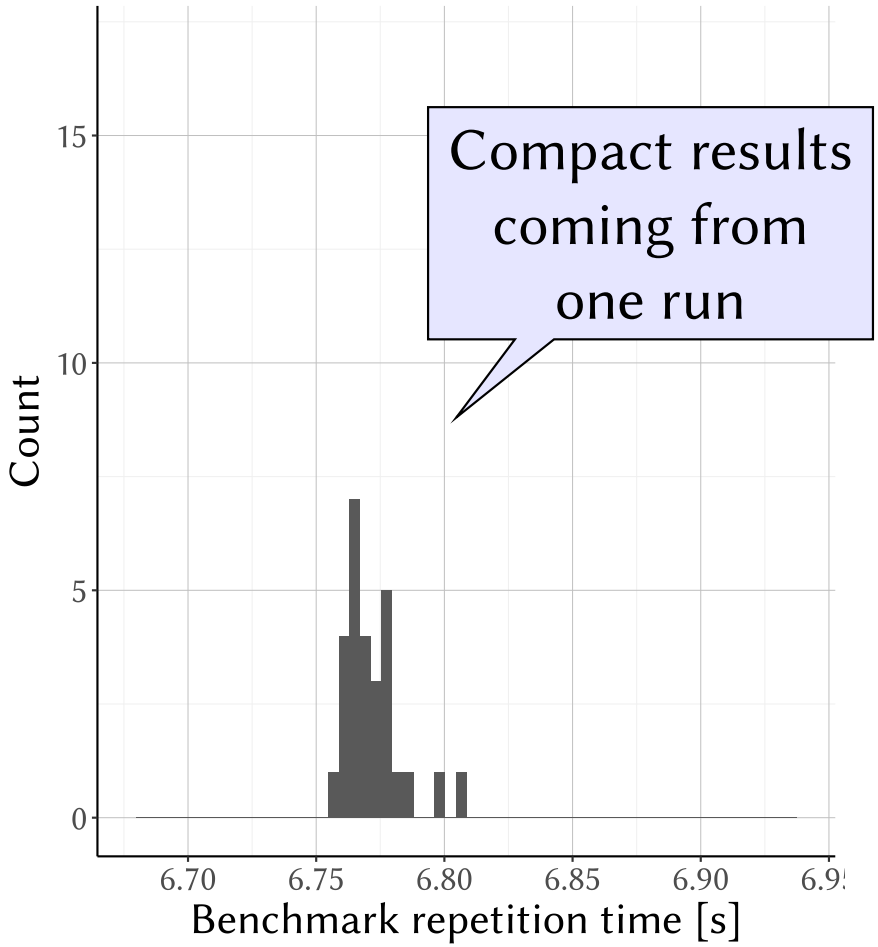
Handling More Runs

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



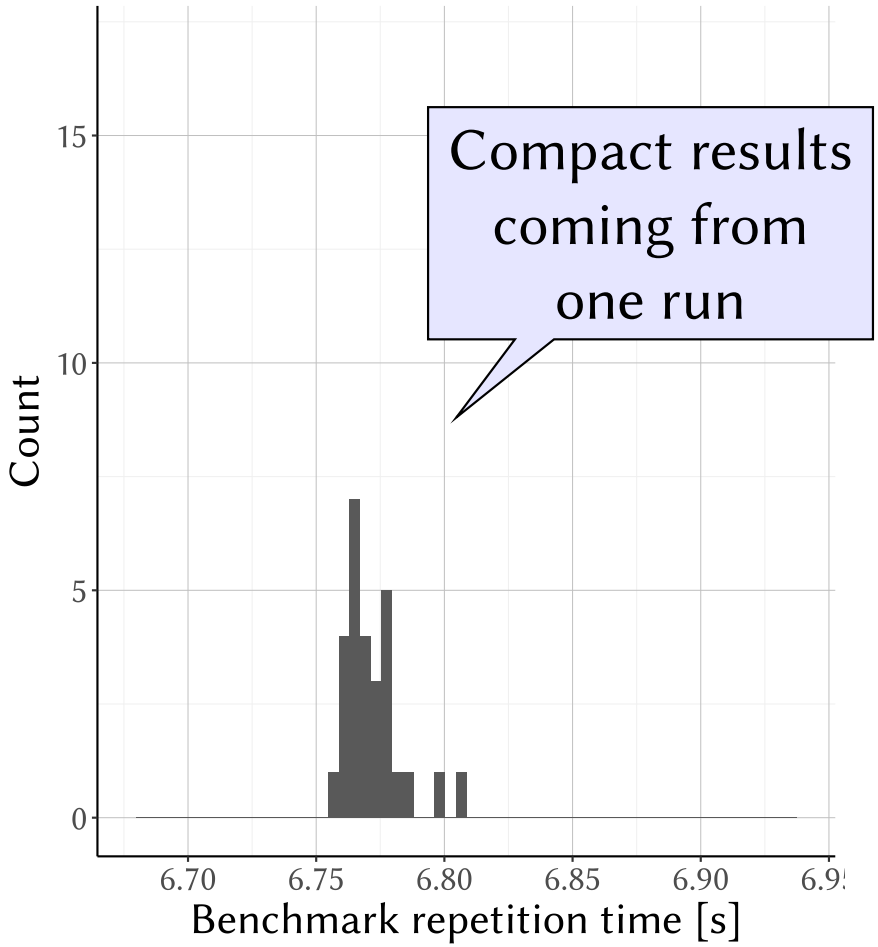
Handling More Runs

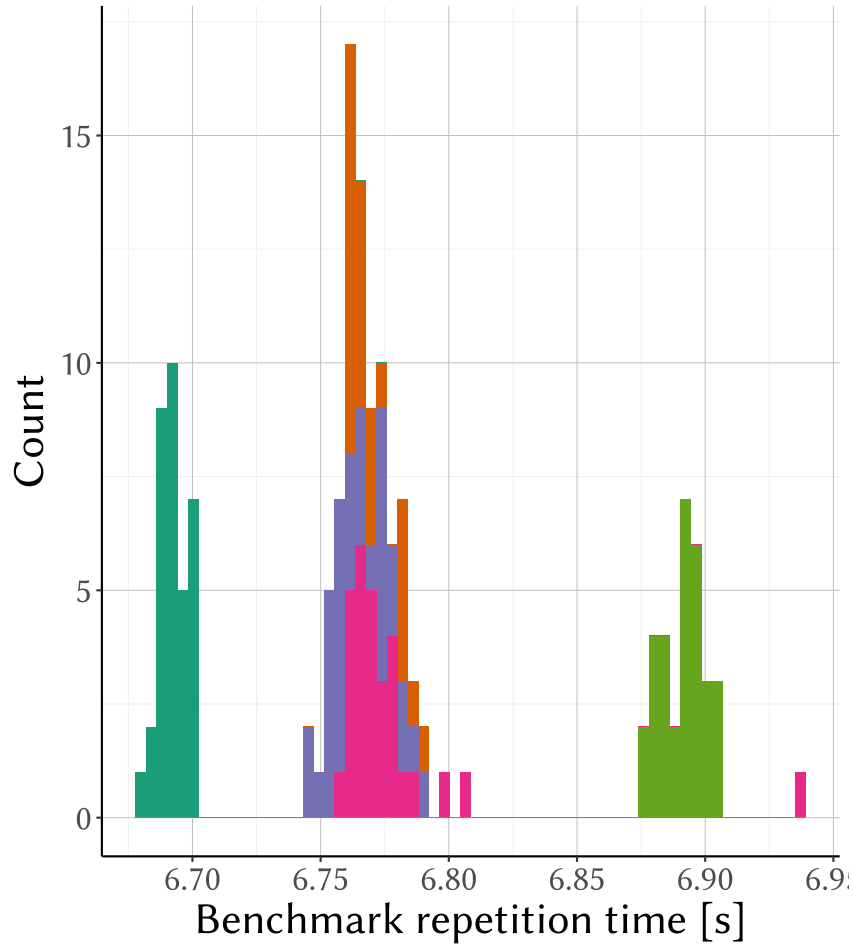
A single benchmark run does not really tell the whole story ...



Handling More Runs

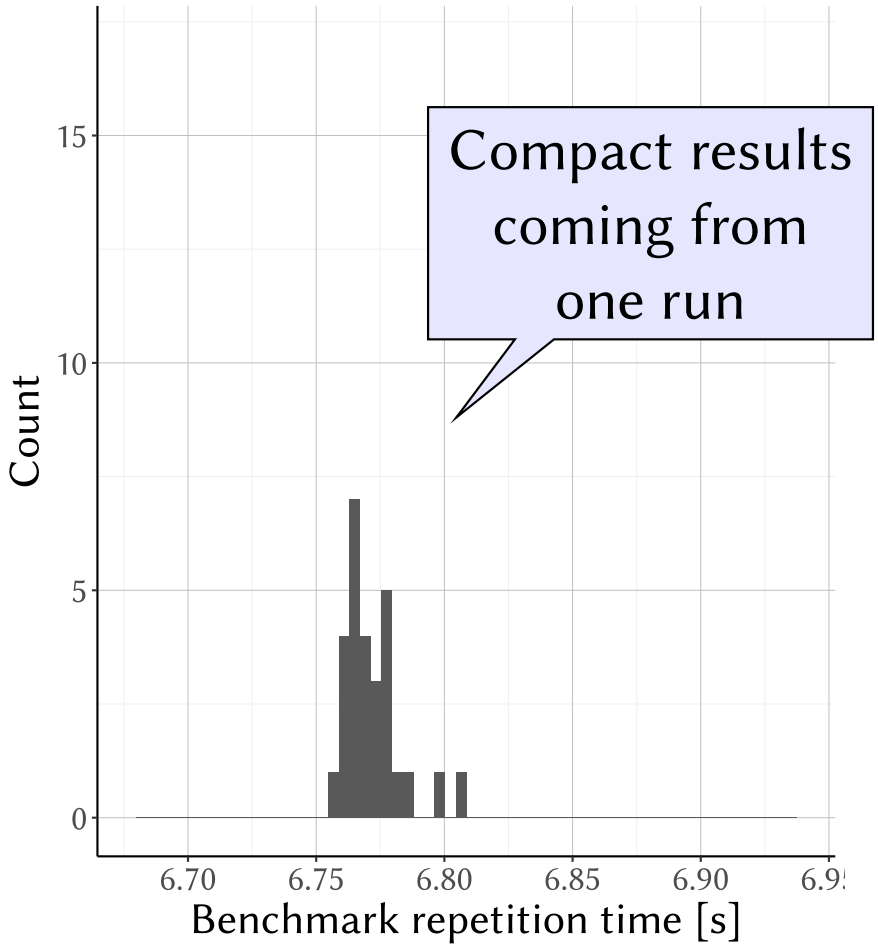
A single benchmark run does not really tell the whole story ...

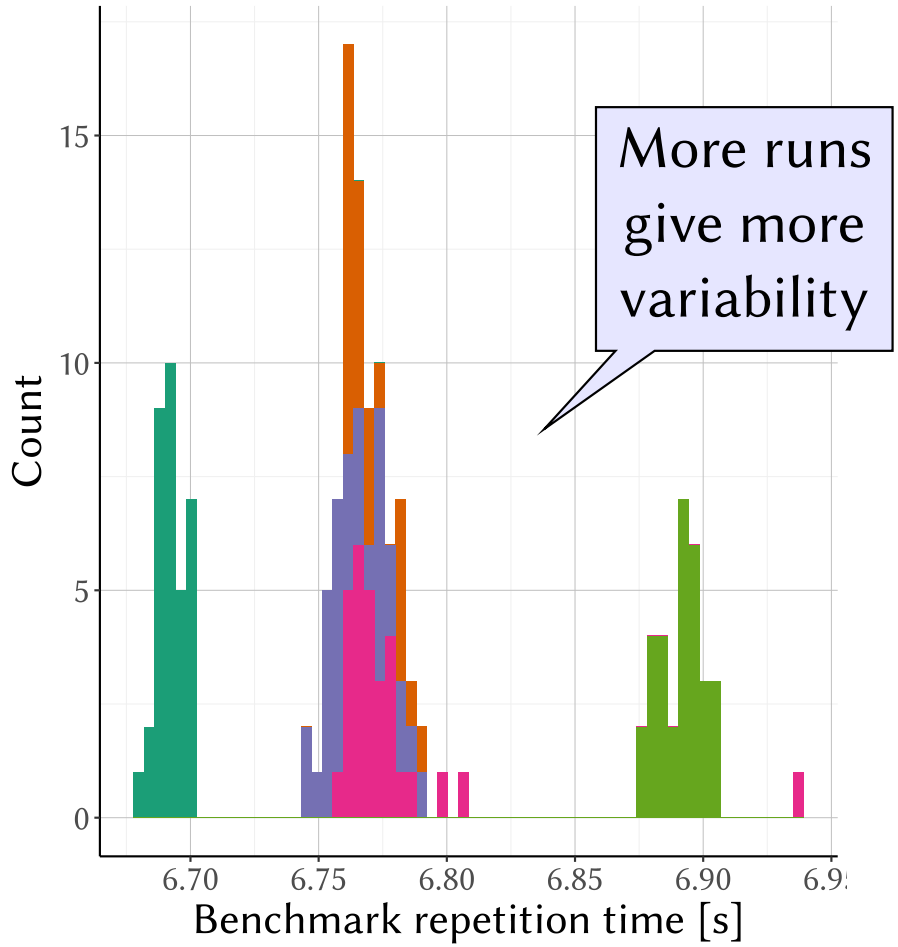




Handling More Runs

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	15	99+	86	99+
als	6	7	99+	99+
chi-sqr	99+	99+	99+	99+
db-shot	99+	99+	56	39
dec-tre	99+	55	99+	99+
dotty	13	16	21	8
fin-chi	99+	99+	99+	99+
fin-htt	25	21	19	24
fj-kmns	70	6	23	69
fut-gen	99+	99+	99+	99+
gauss	99+	99+	99+	99+
log-reg	10	11	21	40
mne	99+	99+	99+	99+
mov-len	5	8	10	4
nai-bay	10	4	99+	99+
neo-ana	99+	99+	100	99+
pg-rank	99+	99+	99+	62
par-mne	99+	84	99+	38
philos	99+	99+	99+	99+
reactr	36	42	99+	99+

rx-scrb	49	65	26	19
sc-doku	99+	99+	99+	99+
sc-kmns	8	5	27	19
sc-stmb	93	68	99+	99+
scrb	99+	99+	99+	99+

ScalaBench (with DaCapo)				
bench	C8	C11	E8	E11
appar-d	99+	99+	27	41
avror-l	8	7	18	7
batik-s	2	1	2	1
eclps-s	10		11	
factr-d	99+	99+	99+	99+
fop-d	17	16	10	25
h2-d	24	32	33	87
jythn-l	31	99+	44	70
kiama-d	39	51	46	18
luidx-d	62	50	23	27
lusrc-l	42	30	27	11
pmd-l	32	61	99+	14
scc-l	99+	99+	23	20
scdoc-l	99+	20	46	19
scp-l	10	19	52	96

scrfm-h	33	13	44	34
scxb-h	99+	99+	99+	99+
specs-l	12	5	11	8
sunfl-l	6	16	99+	18
tmt-d	8	9	19	9
trdb-d	17	26	18	25
trds-l	7	5	3	5
xalan-l	35	26	28	23

SPECjvm2008 (modified)				
bench	C8	C11	E8	E11
cmp.cmp	8		5	
cmp.sun	5		16	
compr	4	99+	15	16
cry.aes	13	21	99+	9
cry.rsa	11	9	6	7
cry.sgn	9	13	5	14
derby	28	8	35	70
mpega	1	1	1	2
sci.ffl	99+	99+	99+	99+
sci.lul	1	1	1	1
sci.mtc	12	6	99+	1
sci.sol	1	1	1	1

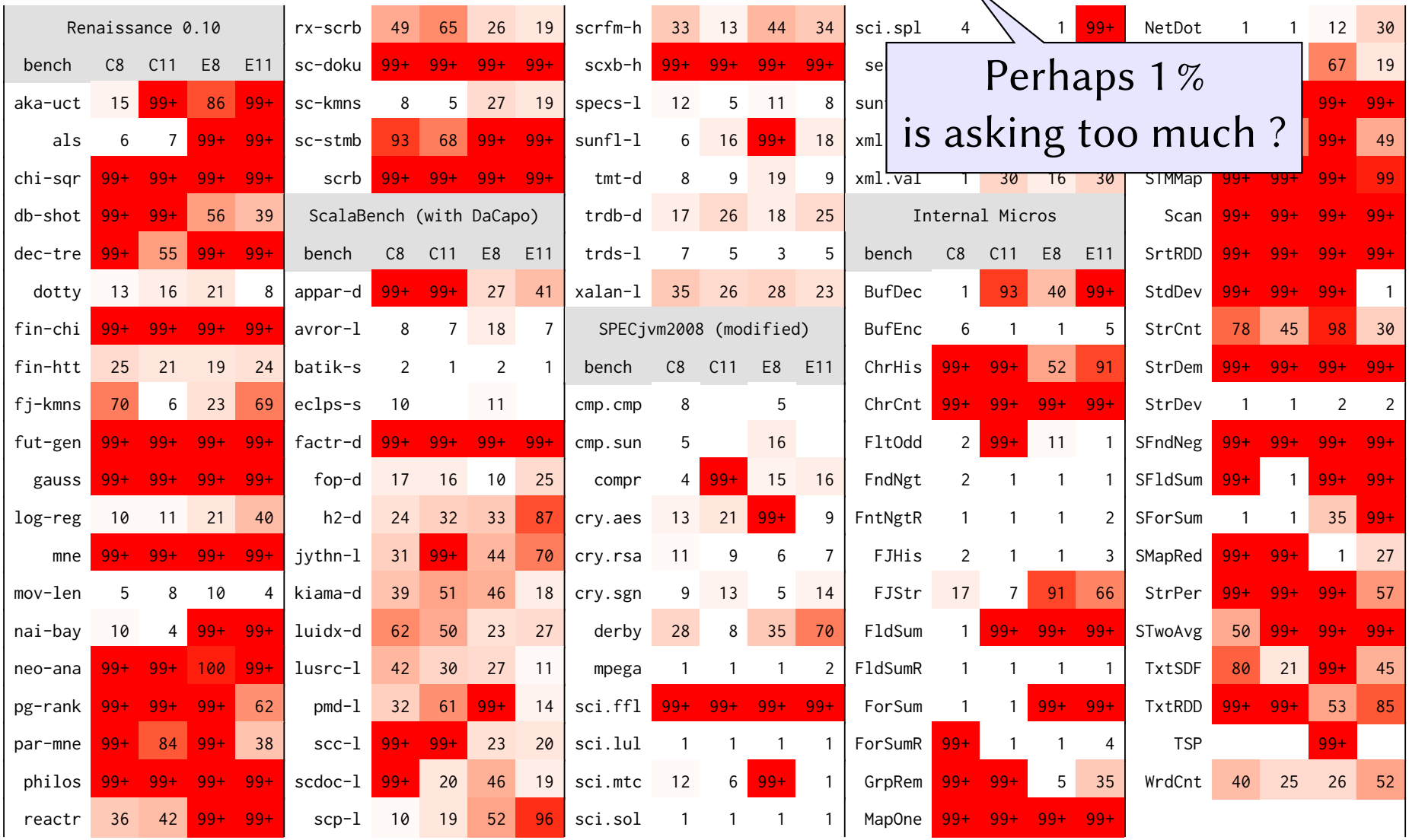
sci.spl	4	9	1	99+
serial	14	23	99+	99+
sunflow	9	13	7	3
xml.trn	10	7	9	7
xml.val	1	30	16	30

Internal Micros				
bench	C8	C11	E8	E11
BufDec	1	93	40	99+
BufEnc	6	1	1	5
ChrHis	99+	99+	52	91
ChrCnt	99+	99+	99+	99+
FltOdd	2	99+	11	1
FndNgt	2	1	1	1
FntNgtR	1	1	1	2
FJHis	2	1	1	3
FJStr	17	7	91	66
FldSum	1	99+	99+	99+
FldSumR	1	1	1	1
ForSum	1	1	99+	99+
ForSumR	99+	1	1	4
GrpRem	99+	99+	5	35
MapOne	99+	99+	99+	99+

NetDot	1	1	12	30
NetEig	1	1	67	19
Reduce	72	99+	99+	99+
STMLst	99+	70	99+	49
STMMap	99+	99+	99+	99
Scan	99+	99+	99+	99+
SrtRDD	99+	99+	99+	99+
StdDev	99+	99+	99+	1
StrCnt	78	45	98	30
StrDem	99+	99+	99+	99+
StrDev	1	1	2	2
SFndNeg	99+	99+	99+	99+
SFldSum	99+	1	99+	99+
SForSum	1	1	35	99+
SMapRed	99+	99+	1	27
StrPer	99+	99+	99+	57
STwoAvg	50	99+	99+	99+
TxtSDF	80	21	99+	45
TxtRDD	99+	99+	53	85
TSP			99+	
WrdCnt	40	25	26	52

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	15	99+	86	99+
als	6	7	99+	99+
chi-sqr	99+	99+	99+	99+
db-shot	99+	99+	56	39
dec-tre	99+	55	99+	99+
dotty	13	16	21	8
fin-chi	99+	99+	99+	99+
fin-htt	25	21	19	24
fj-kmns	70	6	23	69
fut-gen	99+	99+	99+	99+
gauss	99+	99+	99+	99+
log-reg	10	11	21	40
mne	99+	99+	99+	99+
mov-len	5	8	10	4
nai-bay	10	4	99+	99+
neo-ana	99+	99+	100	99+
pg-rank	99+	99+	99+	62
par-mne	99+	84	99+	38
philos	99+	99+	99+	99+
reactr	36	42	99+	99+

rx-scrb	49	65	26	19
sc-doku	99+	99+	99+	99+
sc-kmns	8	5	27	19
sc-stmb	93	68	99+	99+
scrb	99+	99+	99+	99+

ScalaBench (with DaCapo)				
bench	C8	C11	E8	E11
appar-d	99+	99+	27	41
avror-l	8	7	18	7
batik-s	2	1	2	1
eclps-s	10		11	
factr-d	99+	99+	99+	99+
fop-d	17	16	10	25
h2-d	24	32	33	87
jythn-l	31	99+	44	70
kiama-d	39	51	46	18
luidx-d	62	50	23	27
lusrc-l	42	30	27	11
pmd-l	32	61	99+	14
scc-l	99+	99+	23	20
scdoc-l	99+	20	46	19
scp-l	10	19	52	96

scrfm-h	33	13	44	34
scxb-h	99+	99+	99+	99+
specs-l	12	5	11	8
sunfl-l	6	16	99+	18
tmt-d	8	9	19	9
trdb-d	17	26	18	25
trds-l	7	5	3	5
xalan-l	35	26	28	23

SPECjvm2008 (modified)				
bench	C8	C11	E8	E11
cmp.cmp	8		5	
cmp.sun	5		16	
compr	4	99+	15	16
cry.aes	13	21	99+	9
cry.rsa	11	9	6	7
cry.sgn	9	13	5	14
derby	28	8	35	70
mpega	1	1	1	2
sci.ffl	99+	99+	99+	99+
sci.lul	1	1	1	1
sci.mtc	12	6	99+	1
sci.sol	1	1	1	1

sci.spl	4		1	99+
se				
sun				
xml				
xml.val	1	30	16	30

Internal Micros				
bench	C8	C11	E8	E11
BufDec	1	93	40	99+
BufEnc	6	1	1	5
ChrHis	99+	99+	52	91
ChrCnt	99+	99+	99+	99+
FltOdd	2	99+	11	1
FndNgt	2	1	1	1
FntNgtR	1	1	1	2
FJHis	2	1	1	3
FJStr	17	7	91	66
FldSum	1	99+	99+	99+
FldSumR	1	1	1	1
ForSum	1	1	99+	99+
ForSumR	99+	1	1	4
GrpRem	99+	99+	5	35
MapOne	99+	99+	99+	99+

NetDot	1	1	12	30
			67	19
			99+	99+
			99+	49
STMMap	99+	99+	99+	99
Scan	99+	99+	99+	99+
SrtRDD	99+	99+	99+	99+
StdDev	99+	99+	99+	1
StrCnt	78	45	98	30
StrDem	99+	99+	99+	99+
StrDev	1	1	2	2
SFndNeg	99+	99+	99+	99+
SFldSum	99+	1	99+	99+
SForSum	1	1	35	99+
SMapRed	99+	99+	1	27
StrPer	99+	99+	99+	57
STwoAvg	50	99+	99+	99+
TxtSDF	80	21	99+	45
TxtRDD	99+	99+	53	85
TSP			99+	
WrdCnt	40	25	26	52

How Many Runs Needed ...

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

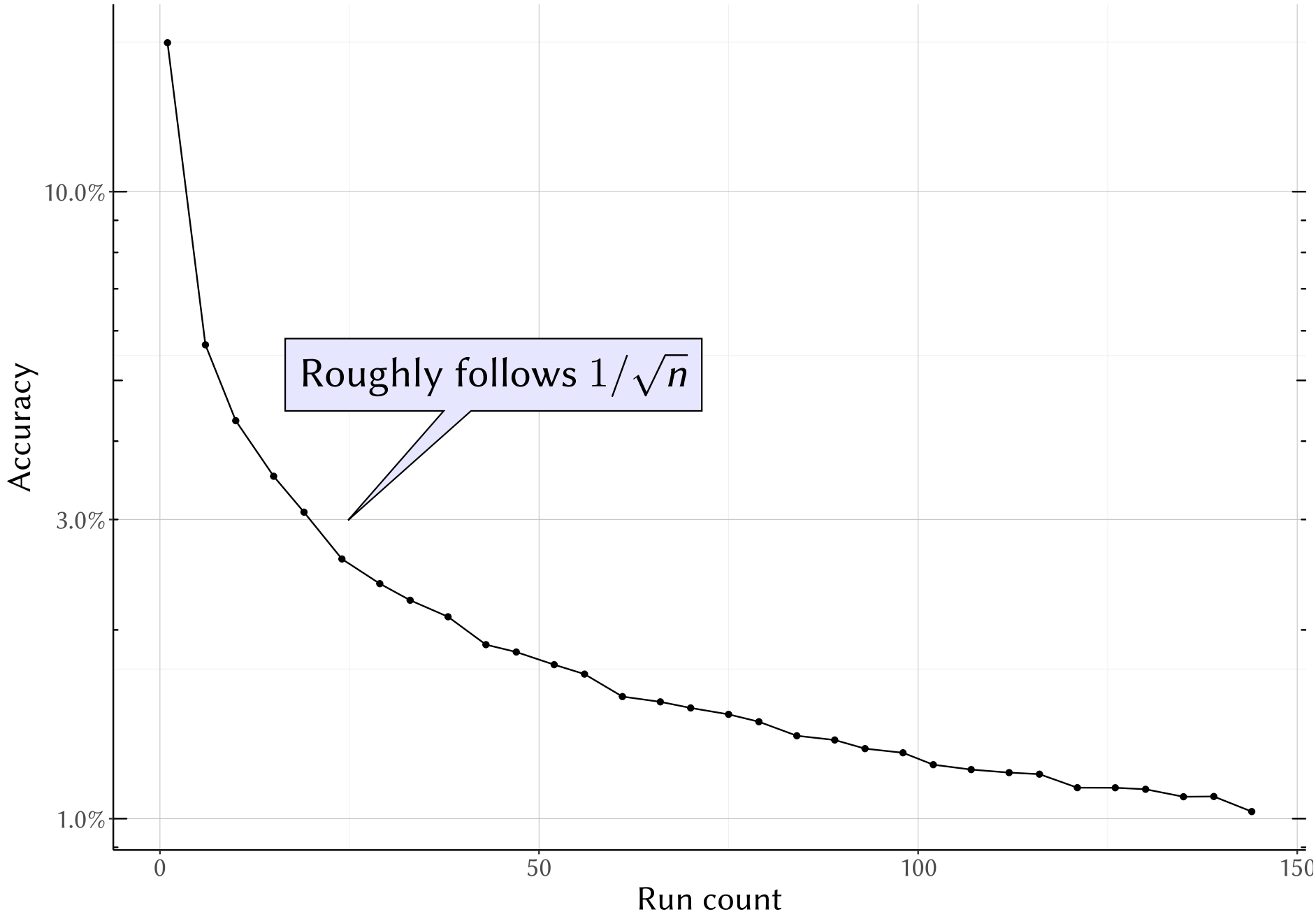
Renaissance 0.10				
bench	C8	C11	E8	E11
aka-uct	1	4	3	4
als	1	2	7	14
chi-sqr	23	22	36	26
db-shot	7	6	2	1
dec-tre	11	1	6	7
dotty	1	1	1	1
fin-chi	5	21	26	6
fin-htt	1	1	1	1
fj-kmns	1	3	2	1
fut-gen	6	6	3	8
gauss	25	13	99+	99+
log-reg	6	8	2	2
mne	7	13	29	12
mov-len	1	1	1	1
nai-bay	1	1	60	100
neo-ana	41	8	10	14
pg-rank	7	5	5	2
par-mne	8	5	99+	1
philos	10	99+	14	38
reactr	2	1	23	10
rx-scrb	2	2	1	1
sc-doku	67	18	99+	99+
sc-kmns	2	1	1	1
sc-stmb	2	2	4	6
scrb	20	10	25	42

ScalaBench (with DaCapo)				
bench	C8	C11	E8	E11
appar-d	99+	99+	3	2
avror-l	2	1	1	1
batik-s	1	1	1	1
eclps-s	2		2	
factr-d	6	7	38	59
fop-d	1	3	1	1
h2-d	1	2	1	2
jythn-l	3	9	1	3
kiama-d	1	6	2	1
luidx-d	1	1	1	2
lusrc-l	1	1	3	1
pmd-l	1	2	13	1
scc-l	5	11	1	1
scdoc-l	4	1	1	1
scp-l	1	1	1	3
scrfm-h	2	1	1	1
scxb-h	8	6	25	99+
specs-l	1	1	3	1
sunfl-l	1	1	2	1
tmt-d	1	1	2	1
trdb-d	1	3	1	1
trds-l	3	1	1	1
xalan-l	1	1	4	1

SPECjvm2008 (modified)				
bench	C8	C11	E8	E11
cmp.cmp	1		1	
cmp.sun	1		4	
compr	1	3	1	2
cry.aes	1	1	11	4
cry.rsa	1	1	1	1
cry.sgn	1	1	1	14
derby	2	1	1	2
mpega	1	1	1	1
sci.ffl	21	14	33	7
sci.lul	1	1	1	1
sci.mtc	1	1	12	1
sci.sol	1	1	1	1
sci.spl	1	1	1	99+
serial	2	8	3	13
sunflow	1	1	1	1
xml.trn	1	1	1	1
xml.val	1	3	1	3

Internal Micros				
bench	C8	C11	E8	E11
BufDec	1	5	8	2
BufEnc	1	1	1	5
ChrHis	4	10	4	3
ChrCnt	11	7	3	5
FltOdd	1	45	6	1
FndNgt	2	1	1	1
FntNgtR	1	1	1	1
FJHis	1	1	1	3
FJStr	1	5	3	2
FldSum	1	3	73	70
FldSumR	1	1	1	1
ForSum	1	1	81	80
ForSumR	10	1	1	4
GrpRem	7	7	4	9
MapOne	14	16	99+	99+
NetDot	1	1	12	30
NetEig	1	1	2	4
Reduce	14	11	8	15
STMLst	6	21	8	1
STMMap	18	99+	24	4
Scan	9	14	34	8
SrtRDD	4	7	5	19
StdDev	45	99+	99+	1
StrCnt	3	9	7	1
StrDem	99+	26	99+	51
StrDev	1	1	2	2
SFndNeg	11	9	18	12
SFldSum	34	1	99+	99+
SForSum	1	1	21	44
SMapRed	67	57	1	1
StrPer	13	99+	99+	1
STwoAvg	25	40	99+	99+
TxtSDF	3	1	8	10
TxtRDD	11	10	1	8
TSP			72	
WrdCnt	1	5	2	3

How Accuracy Relates To Run Count ?



Take Away So Far ...

Running benchmarks only once may not be enough

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

Aiming for excessive accuracy backfires quickly

Reasonable accuracy is a function of more than just the benchmark

- Tooling should consider benchmarks together with platforms
- Not yet sure how often relevant parameters tend to change

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	15	16	21
als	6	4	4
chi-sqr	99+	99+	99+
db-shot	99+	99+	99+
dec-tre	99+	99+	99+
dotty	13	14	6
fin-chi	99+	99+	99+
fin-htt	25	49	15
fj-kmns	70	81	60
fut-gen	99+	99+	99+
gauss	99+	99+	99+
log-reg	10	11	2
mne	99+	99+	99+
mov-len	5	8	9
nai-bay	10	9	99
neo-ana	99+	99+	99+
pg-rank	99+	99+	99+
par-mne	99+	99+	99+
philos	99+	99+	50
reactr	36	85	48
rx-scrb	49	46	25
sc-doku	99+	99+	99+
sc-kmns	8	8	7
sc-stmb	93	99+	99+
scrb	99+	99+	99+

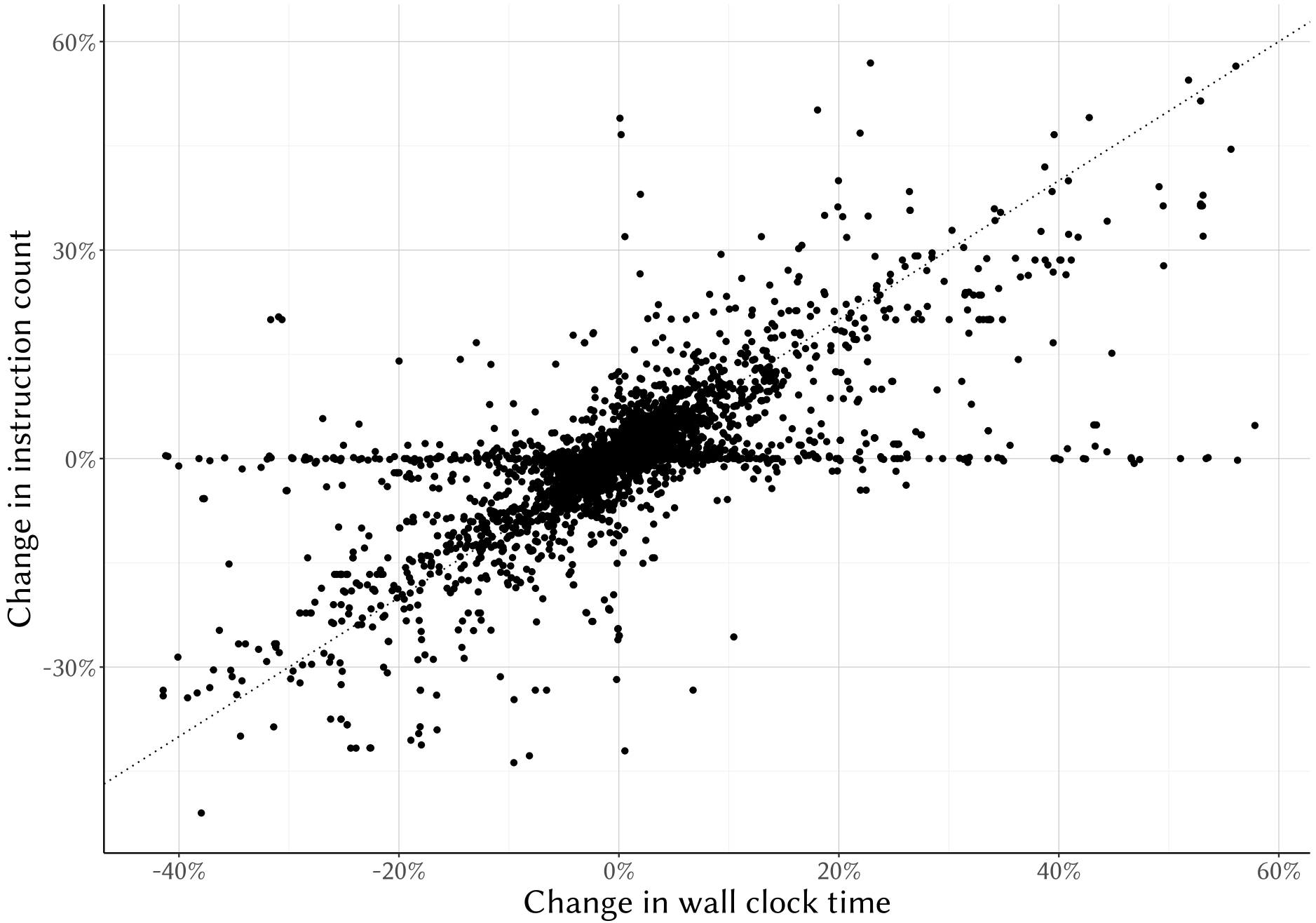
ScalaBench (with DaCapo)			
bench	time	clk	ins
appar-d	99+	99+	99+
avror-l	8	32	88
batik-s	2	2	1
eclps-s	10	12	1
factr-d	99+	99+	99+
fop-d	17	17	6
h2-d	24	10	12
jythn-l	31	31	9
kiama-d	39	66	51
luidx-d	62	7	5
lusrc-l	42	54	29
pmd-l	32	16	11
scc-l	99+	99+	99+
scdoc-l	99+	99+	99+
scp-l	10	65	56
scrfm-h	33	69	75
scxb-h	99+	99+	39
specs-l	12	27	14
sunfl-l	6	6	8
tmt-d	8	14	45
trdb-d	17	99+	99+
trds-l	7	12	7
xalan-l	35	99+	99+

SPECjvm2008 (modified)			
bench	time	clk	ins
cmp.cmp	8	8	8
cmp.sun	5	5	11
compr	4	4	1
cry.aes	13	13	1
cry.rsa	11	11	3
cry.sgn	9	9	18
derby	28	28	5
mpega	1	1	1
sci.ffl	99+	99+	1
sci.lul	1	1	1
sci.mtc	12	12	23
sci.sol	1	1	1
sci.spl	4	4	23
serial	14	14	2
sunflow	9	9	11
xml.trn	10	11	1
xml.val	1	3	1

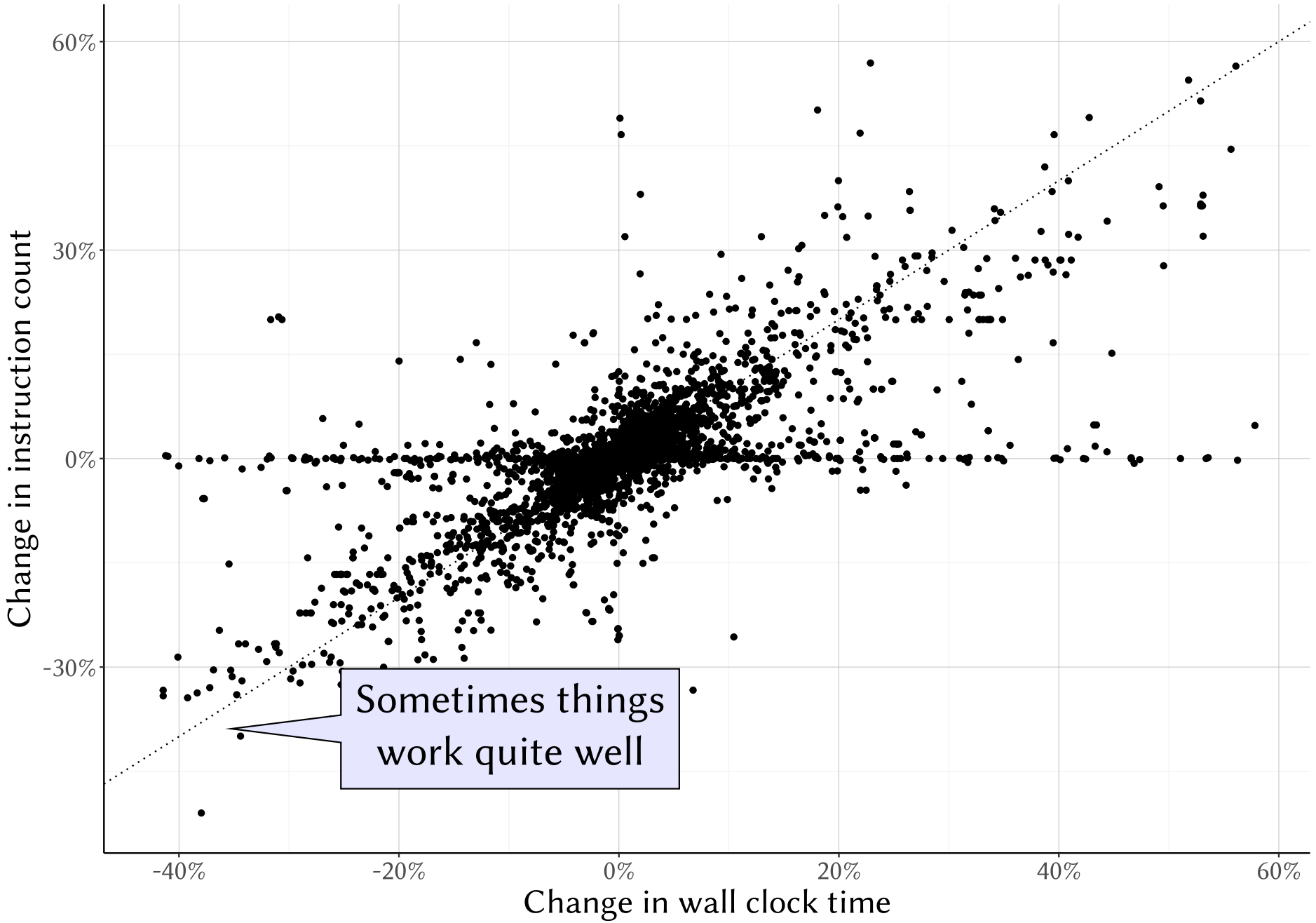
Internal Micros			
bench	time	clk	ins
BufDec	1	1	1
BufEnc	6	6	2
ChrHis	99+	99+	55
ChrCnt	99+	99+	50
FltOdd	2	2	1
FndNgt	2	1	1
FntNgtR	1	1	1
FJHis	2	2	3
FJStr	17	23	11
FldSum	1	1	1
FldSumR	1	1	1
ForSum	1	1	1
ForSumR	99+	99+	1
GrpRem	99+	99+	99+
MapOne	99+	99+	99+
NetDot	1	1	1
NetEig	1	1	1
Reduce	72	99+	60
STMLst	99+	99+	99+
STMMap	99+	99+	99+
Scan	99+	99+	32
SrtRDD	99+	99+	25
StdDev	99+	99+	99+
StrCnt	78	99+	63
StrDem	99+	99+	99+
StrDev	1	1	9
SFndNeg	99+	99+	99+
SFldSum	99+	99+	99+
SForSum	1	1	1
SMapRed	99+	99+	99+
StrPer	99+	99+	34
STwoAvg	50	51	38
TxtSDF	80	99+	29
TxtRDD	99+	99+	34
WrdCnt	40	65	32

time - wall clock time clk - thread clock time ins - instruction count

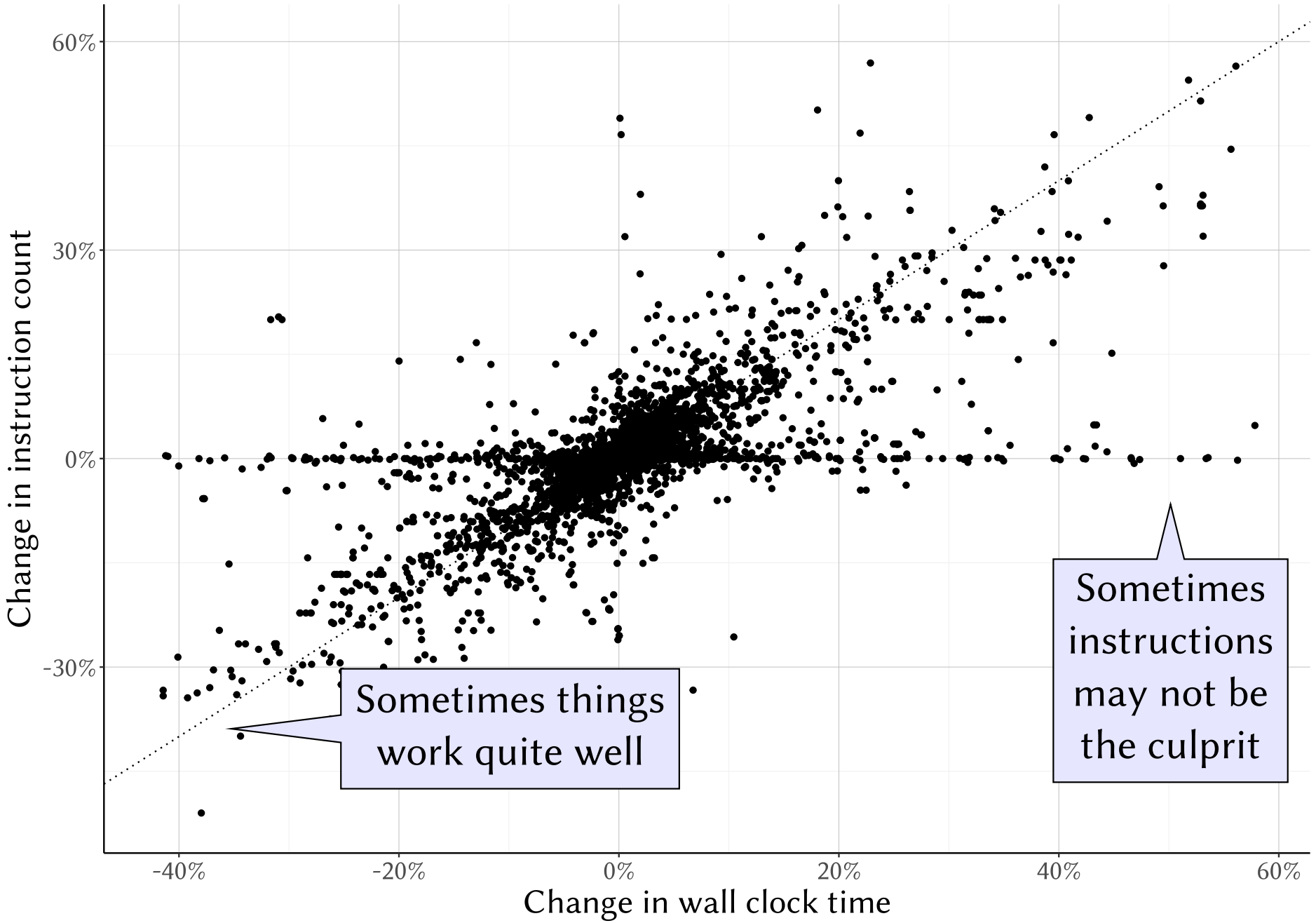
Different Metrics Not Always In Sync



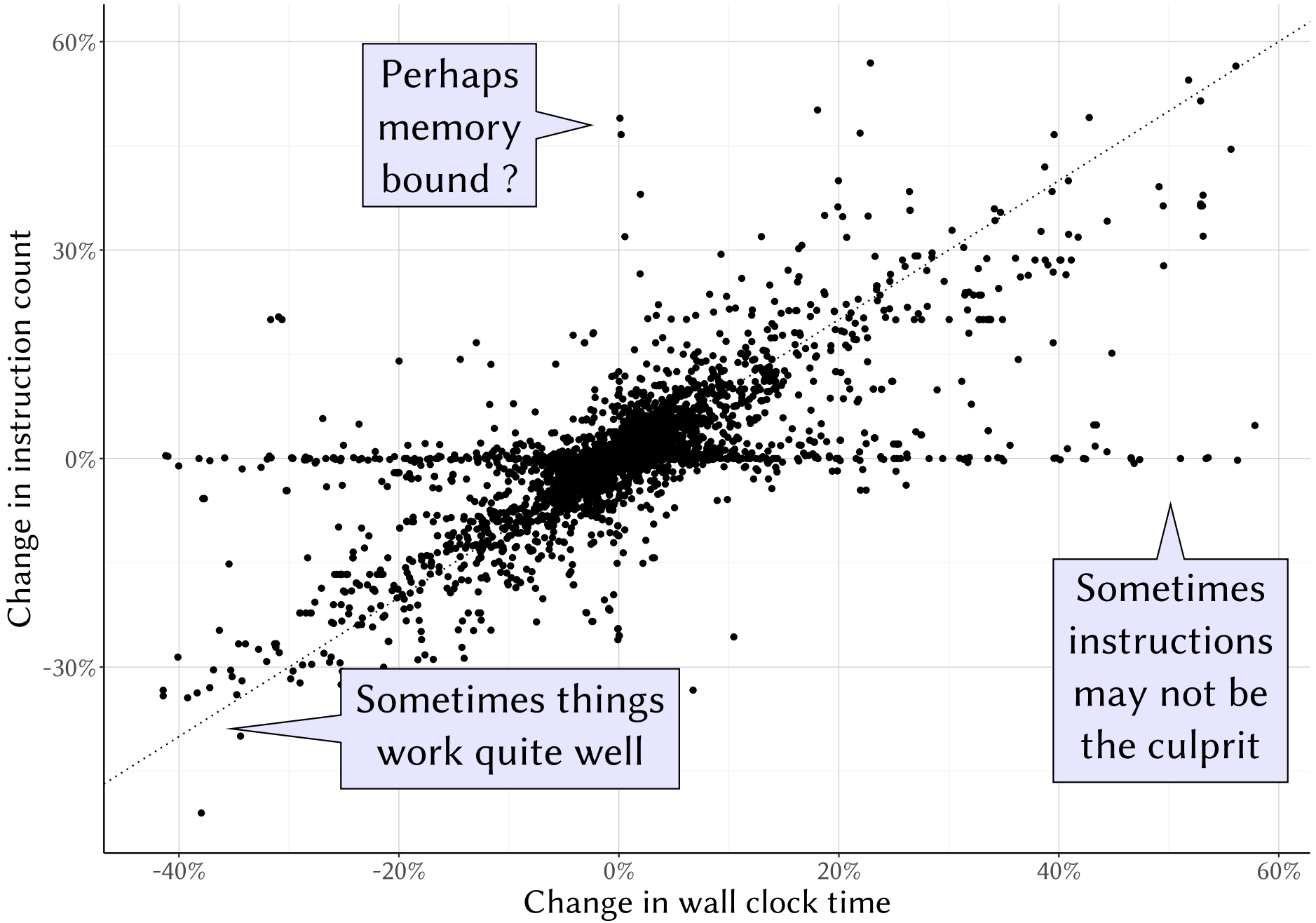
Different Metrics Not Always In Sync



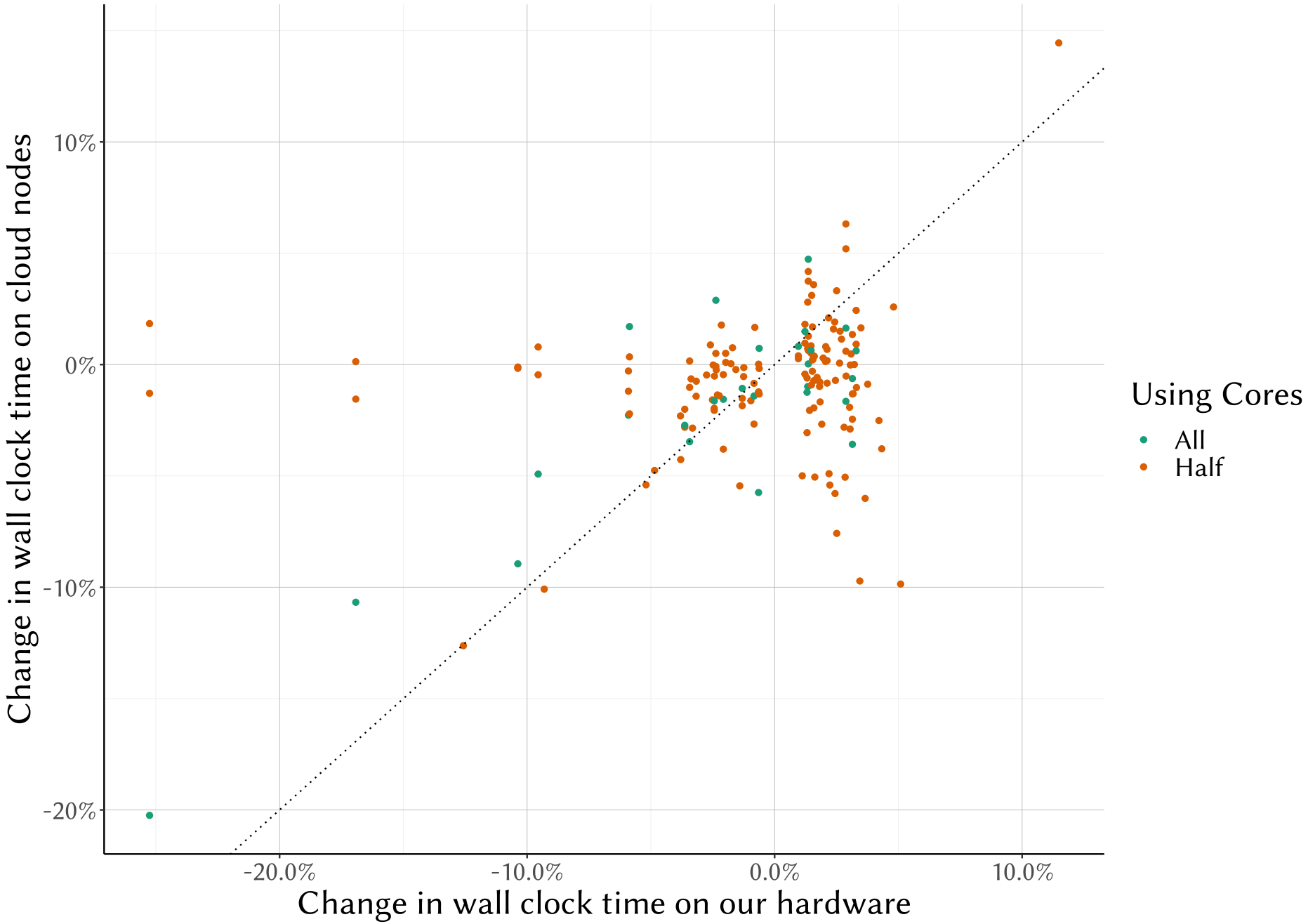
Different Metrics Not Always In Sync



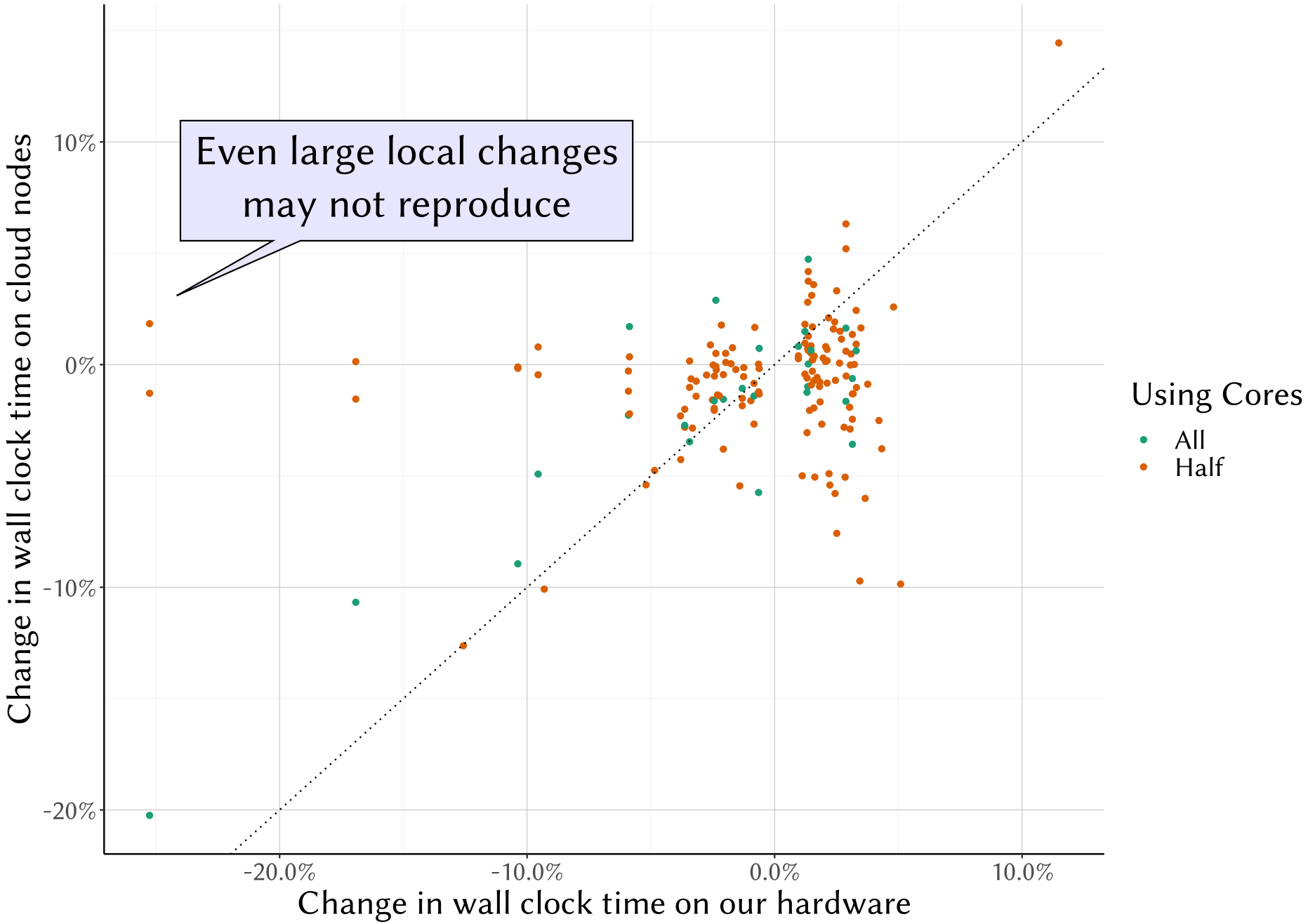
Different Metrics Not Always In Sync



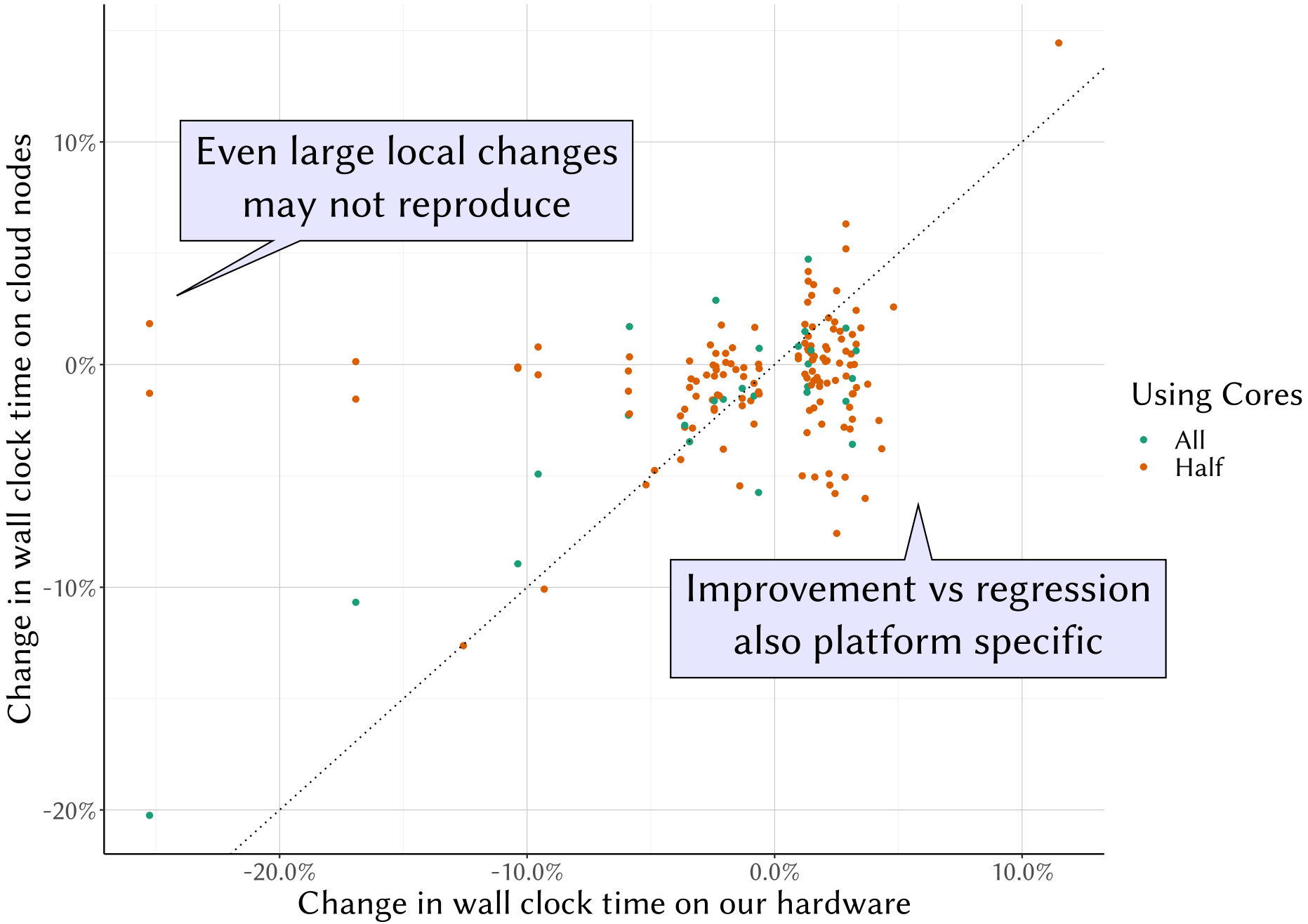
Wall Clock Time Changes Not Always Portable



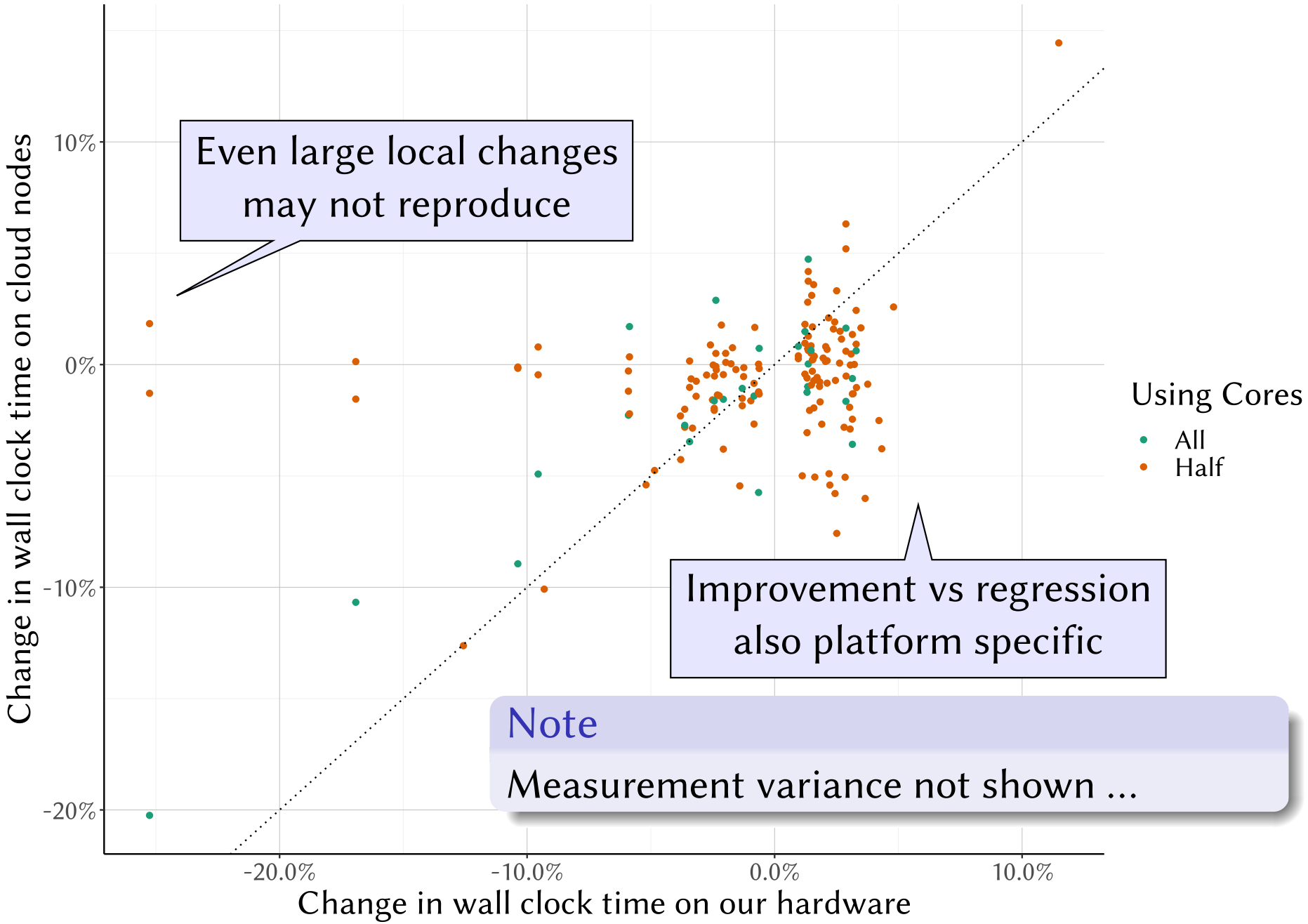
Wall Clock Time Changes Not Always Portable



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

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 whether performance regression should be addressed

- Especially with benchmarks that
  do not represent application performance
- Effort needed to investigate reasons is not very predictable

Thank You !

https://d3s.mff.cuni.cz

Our work is kindly sponsored by Oracle Labs.

Thank You !

Do not treat all benchmarks the same ...

... using similar run sizes or expecting similar accuracy is not a good idea

https://d3s.mff.cuni.cz

Our work is kindly sponsored by Oracle Labs.

Thank You !

Do not treat all benchmarks the same ...

... using similar run sizes or expecting similar accuracy is not a good idea

Microbenchmarks should get special treatment ...

... good for seeing specific changes but bad for judging practical impact

https://d3s.mff.cuni.cz

Our work is kindly sponsored by Oracle Labs.

Thank You !

Do not treat all benchmarks the same ...

... using similar run sizes or expecting similar accuracy is not a good idea

Microbenchmarks should get special treatment ...

... good for seeing specific changes but bad for judging practical impact

Evaluation cannot stay with developers only ...

... users determine important workloads

https://d3s.mff.cuni.cz

Our work is kindly sponsored by Oracle Labs.

Thank You !

Do not treat all benchmarks the same ...

... using similar run sizes or expecting similar accuracy is not a good idea

Microbenchmarks should get special treatment ...

... good for seeing specific changes but bad for judging practical impact

Evaluation cannot stay with developers only ...

... users determine important workloads

Contribute to Renaissance ...

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

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

Our work is kindly sponsored by Oracle Labs.