Analysis of Branch Misses in Quicksort

Sebastian Wild

wild@cs.uni-kl.de

based on joint work with **Conrado Martínez** and **Markus E. Nebel**



04 January 2015

Meeting on Analytic Algorithmics and Combinatorics

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**

```
41          ;
42  i := i + 1
43  a := A[i]
44  IF a < p GOTO 42
45  j := j - 1
46  a := A[j]
47  IF a > p GOTO 45
48          ;
```

- Pipeline stalls are costly

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
        ⋮
i := i + 1
a := A[i]
IF a < p GOTO 42
j := j - 1
a := A[j]
IF a > p GOTO 45
        ⋮
```

⇝ *Pipeline stalls are **costly***

- **Pipeline:** each instruction broken in 4 stages
- **Pipeline:** simpler steps ⇝ shorter CPU cycles
- **Pipeline:** one instruction per cycle (ideal)
- **Pipeline:** ⇝ 4 instructions per cycle

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
41          ⋮
42  i := i + 1
43  a := A[i]
44  IF a < p GOTO 42
45  j := j - 1
46  a := A[j]
47  IF a > p GOTO 45
48          ⋮
```

- each instruction broken in 4 **stages**
- simpler steps ⇝ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

⇝ *Pipeline stalls are **costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**

```
41
42  i := i + 1
43  a := A[i]
44  IF a < p GOTO 42
45  j := j - 1
46  a := A[j]
47  IF a > p GOTO 45
48
```

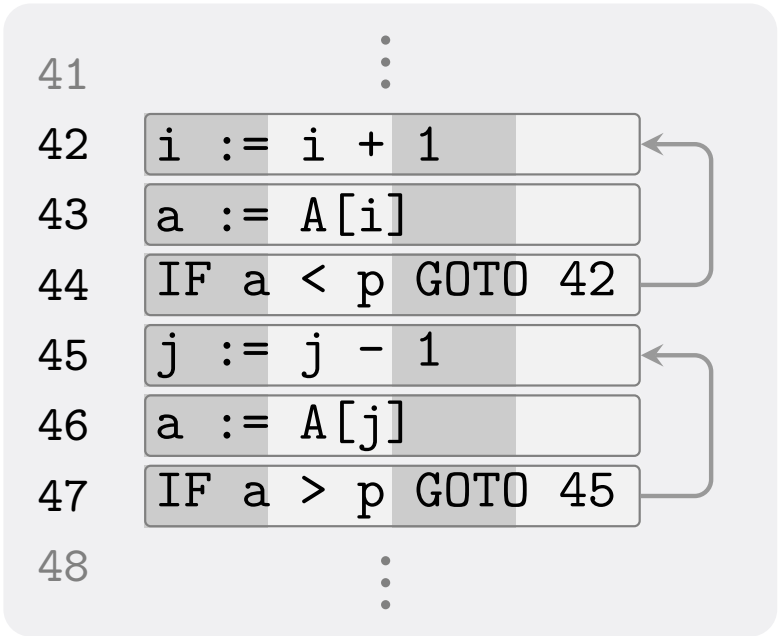


- each instruction broken in 4 **stages**
- simpler steps ⇝ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

⇝ *Pipeline stalls are **costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**





- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

$\rightsquigarrow$ *Pipeline stalls are* ***costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**

```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```



- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

$\rightsquigarrow$ *Pipeline stalls are* ***costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**

```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```



- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

$\rightsquigarrow$ *Pipeline stalls are* ***costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



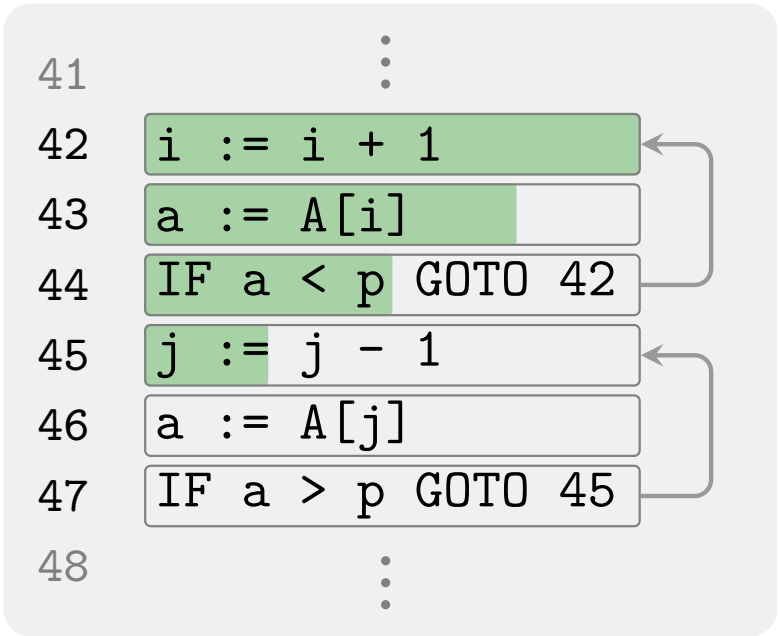
```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```

- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

$\rightsquigarrow$ *Pipeline stalls are **costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**





- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

$\rightsquigarrow$ *Pipeline stalls are* ***costly***

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```

- each instruction broken in 4 **stages**
- simpler steps ⇝ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!

⇝ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```

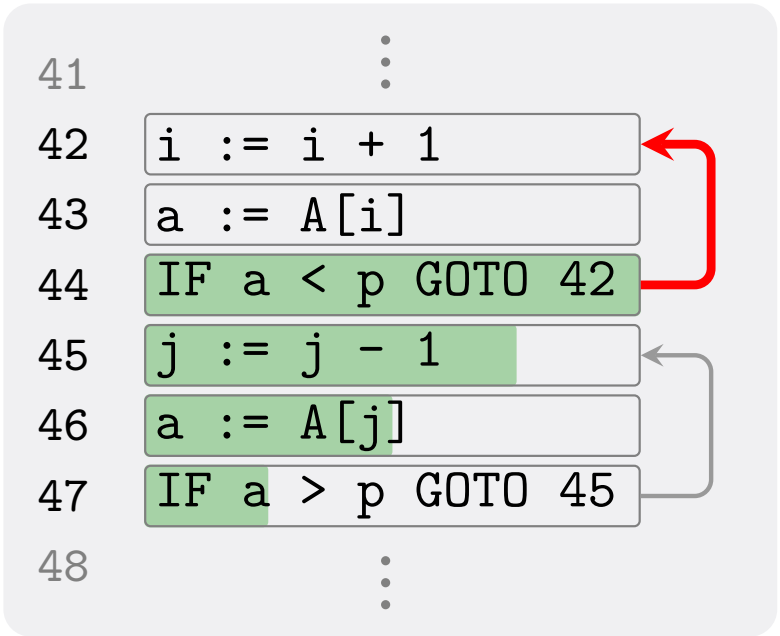
- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**





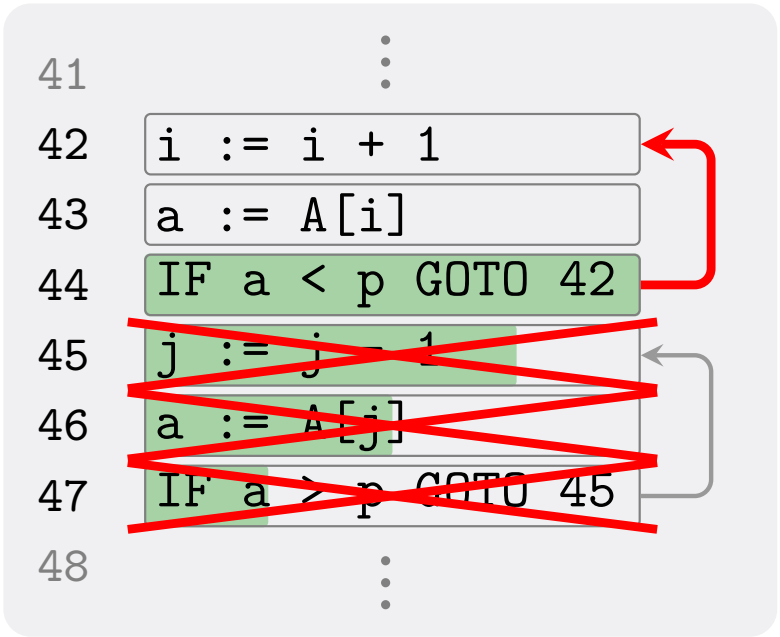
- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**





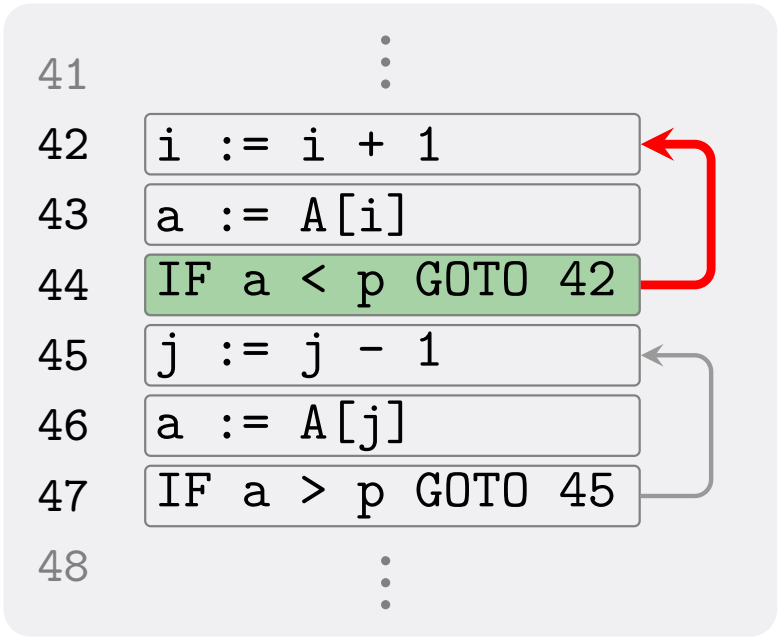
- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**





- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```

- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```

- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Instruction Pipelines

- Computers do **not** execute instructions fully **sequentially**
- Instead they use an *"assembly line"*
- **Example:**



```
41   ⋮
42   i := i + 1
43   a := A[i]
44   IF a < p GOTO 42
45   j := j - 1
46   a := A[j]
47   IF a > p GOTO 45
48   ⋮
```

- each instruction broken in 4 **stages**
- simpler steps $\rightsquigarrow$ shorter CPU cycles
- one instruction per cycle finished ...
- ... except for **branches**!
  1. undo wrong instructions
  2. fill pipeline anew

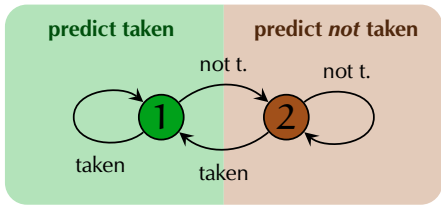
$\rightsquigarrow$ *Pipeline stalls are **costly** ... can we avoid (some of) them?*

Branch Prediction

- We could avoid stalls if we knew whether a branch will be taken or not
- in general not possible $\rightsquigarrow$ prediction with heuristics:
  - Predict same outcome as last time. (**1-bit predictor**)
  - Predict most frequent outcome with finite memory (**2-bit saturating counter**)
  - Flip prediction only after two consecutive errors (**2-bit flip-consecutive**)
  - wilder heuristics exist out there ... not considered here
- prediction can be wrong $\rightsquigarrow$ **branch miss** (BM)

Branch Prediction

- We could avoid stalls if we knew whether a branch will be taken or not
- in general not possible $\rightsquigarrow$ prediction with heuristics:
  - Predict same outcome as last time. (**1-bit predictor**)



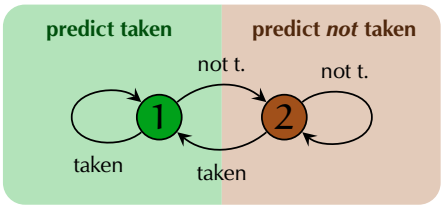
  - Predict most frequent outcome with finite memory (**2-bit saturating counter**)

  - Flip prediction only after two consecutive errors (**2-bit flip-consecutive**)

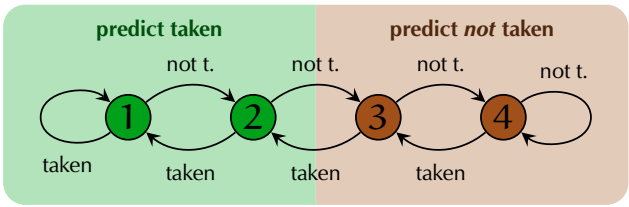
  - wilder heuristics exist out there ... not considered here
- prediction can be wrong $\rightsquigarrow$ **branch miss** (BM)

Branch Prediction

- We could avoid stalls if we knew whether a branch will be taken or not
- in general not possible $\rightsquigarrow$ prediction with heuristics:
  - Predict same outcome as last time. (**1-bit predictor**)



  - Predict most frequent outcome with finite memory (**2-bit saturating counter**)

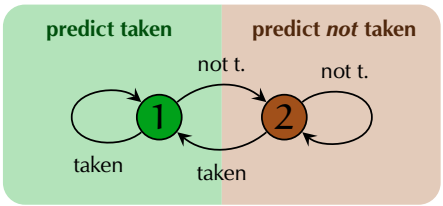


  - Flip prediction only after two consecutive errors (**2-bit flip-consecutive**)

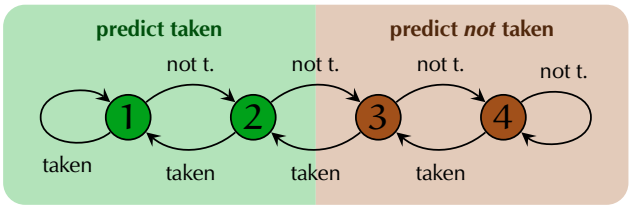
  - wilder heuristics exist out there ... not considered here
- prediction can be wrong $\rightsquigarrow$ **branch miss** (BM)

Branch Prediction

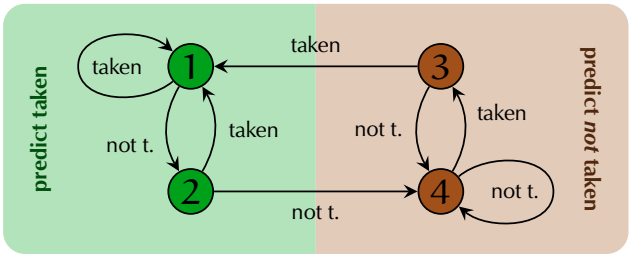
- We could avoid stalls if we knew whether a branch will be taken or not
- in general not possible $\rightsquigarrow$ prediction with heuristics:
  - Predict same outcome as last time. (**1-bit predictor**)



  - Predict most frequent outcome with finite memory (**2-bit saturating counter**)



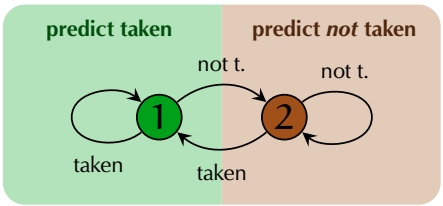
  - Flip prediction only after two consecutive errors (**2-bit flip-consecutive**)



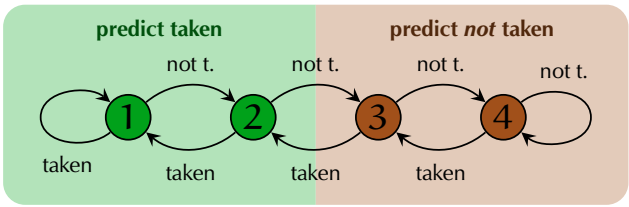
  - wilder heuristics exist out there . . . not considered here
- prediction can be wrong $\rightsquigarrow$ **branch miss** (BM)

Branch Prediction

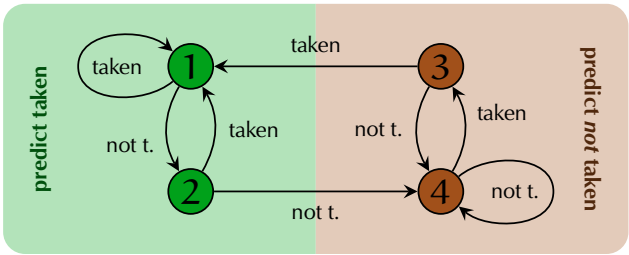
- We could avoid stalls if we knew whether a branch will be taken or not
- in general not possible $\rightsquigarrow$ prediction with heuristics:
  - Predict same outcome as last time. (**1-bit predictor**)



  - Predict most frequent outcome with finite memory (**2-bit saturating counter**)



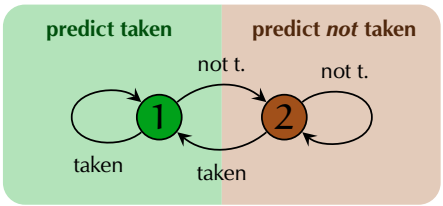
  - Flip prediction only after two consecutive errors (**2-bit flip-consecutive**)



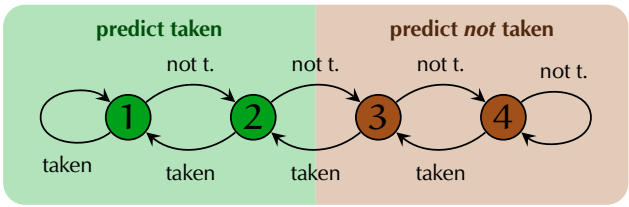
  - wilder heuristics exist out there ... not considered here
- prediction can be wrong $\rightsquigarrow$ **branch miss** (BM)

Branch Prediction

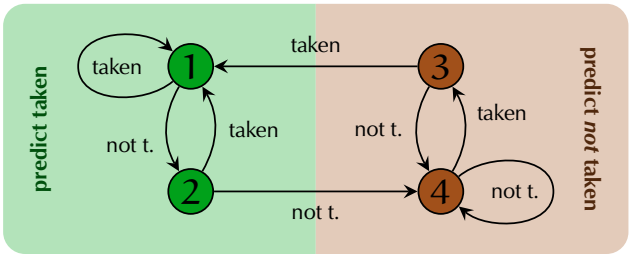
- We could avoid stalls if we knew whether a branch will be taken or not
- in general not possible $\rightsquigarrow$ prediction with heuristics:
  - Predict same outcome as last time. (**1-bit predictor**)



  - Predict most frequent outcome with finite memory (**2-bit saturating counter**)



  - Flip prediction only after two consecutive errors (**2-bit flip-consecutive**)



  - wilder heuristics exist out there ... not considered here
- prediction can be wrong $\rightsquigarrow$ **branch miss** (BM)

Why Should We Care?

- misprediction rates of "typical" programs $< 10\%$

- (Comparison-based) **sorting** is **different**!
  - Branch based on **comparison** result
  - Comparisons reduce **entropy** (uncertainty about input)

$\rightsquigarrow$ The **less** comparisons we use, the **less predictable** they become
  - for classic Quicksort: misprediction rate $\geqslant$ **25 %**
  - with *median-of-3*: $\geqslant$ **31.25 %**

- **Practical Importance** (KALIGOSI & SANDERS, ESA 2006):
  - on Pentium 4 Prescott: very skewed pivot faster than median
  - $\rightsquigarrow$ branch misses **dominated** running time

Why Should We Care?

- misprediction rates of "typical" programs $< 10\%$

- (Comparison-based) **sorting** is **different**!
  - Branch based on **comparison** result
  - Comparisons reduce **entropy** (uncertainty about input)

$\rightsquigarrow$ The **less** comparisons we use, the **less predictable** they become

  - for classic Quicksort: misprediction rate $\geqslant$ **25 %**
  - with *median-of-3:* $\geqslant$ **31.25 %**

- **Practical Importance** (KALIGOSI & SANDERS, ESA 2006):
  - on Pentium 4 Prescott: very skewed pivot faster than median

  $\rightsquigarrow$ branch misses **dominated** running time

Why Should We Care?

- misprediction rates of "typical" programs $< 10\%$

- (Comparison-based) **sorting** is **different**!
  - Branch based on **comparison** result
  - Comparisons reduce **entropy** (uncertainty about input)

$\rightsquigarrow$ The **less** comparisons we use, the **less predictable** they become
  - for classic Quicksort: misprediction rate $\geqslant$ **25 %**
  - with *median-of-3:* $\geqslant$ **31.25 %**

- **Practical Importance** (KALIGOSI & SANDERS, ESA 2006):
  - on Pentium 4 Prescott: very skewed pivot faster than median
  - $\rightsquigarrow$ branch misses **dominated** running time

Why Should We Care?

- misprediction rates of "typical" programs $< 10\%$

- (Comparison-based) **sorting** is **different**!
  - Branch based on **comparison** result
  - Comparisons reduce **entropy** (uncertainty about input)

$\rightsquigarrow$ The **less** comparisons we use, the **less predictable** they become
  - for classic Quicksort: misprediction rate $\geqslant$ **25 %**
  - with *median-of-3:* $\geqslant$ **31.25 %**

- **Practical Importance** (KALIGOSI & SANDERS, ESA 2006):
  - on Pentium 4 Prescott: very skewed pivot faster than median

  $\rightsquigarrow$ branch misses **dominated** running time

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's** ***dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		−10±2%
Comparisons	2	1.9	−5%
Swaps	$0.\overline{3}$	0.6	+80%
Bytecode Instructions	18	21.7	+20.6%
MMIX oops υ	11	13.1	+19.1%
MMIX mems μ	$2.\overline{6}$	2.8	+5%
scanned elements[1] *(≈ cache misses)*	2	1.6	−20%

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's** ***dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		−10±2%
Comparisons	2	1.9	−5%
Swaps	$0.\overline{3}$	0.6	+80%
Bytecode Instructions	18	21.7	+20.6%
MMIX oops υ	11	13.1	+19.1%
MMIX mems μ	$2.\overline{6}$	2.8	+5%
scanned elements[1] *(≈ cache misses)*	2	1.6	−20%

$n \ln n + O(n)$, *average case results*

What about branch misses? Can they explain CQS's superiority?

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's** ***dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		$-10\pm2\%$
Comparisons	2	1.9	-5%
Swaps	$0.\overline{3}$	0.6	$+80\%$
Bytecode Instructions	18	21.7	$+20.6\%$
MMIX oops υ	11	13.1	$+19.1\%$
MMIX mems μ	$2.\overline{6}$	2.8	$+5\%$
scanned elements[1] *(≈ cache misses)*	2	1.6	-20%

$\cdot n \ln n + O(n)$, *average case results*

*What about **branch misses**? Can they **explain** YQS's success? ... stay tuned.*

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's** ***dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		$-10\pm2\%$
Comparisons	2	1.9	-5%
Swaps	$0.\overline{3}$	0.6	$+80\%$
Bytecode Instructions	18	21.7	$+20.6\%$
MMIX oops υ	11	13.1	$+19.1\%$
MMIX mems μ	$2.\overline{6}$	2.8	$+5\%$
scanned elements[1] *(≈ cache misses)*	2	1.6	-20%

$\cdot n \ln n + O(n)$, *average case results*

*What about **branch misses**? Can they **explain** YQS's success? . . . stay tuned.*

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's** ***dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		$-10\pm2\%$
Comparisons	2	1.9	-5%
Swaps	$0.\overline{3}$	0.6	$+80\%$
Bytecode Instructions	18	21.7	$+20.6\%$
MMIX oops υ	11	13.1	$+19.1\%$
MMIX mems μ	$2.\overline{6}$	2.8	$+5\%$
scanned elements[1] *(≈ cache misses)*	2	1.6	-20%

$\cdot n \ln n + O(n)$, *average case results*

*What about **branch misses**? Can they **explain** YQS's success? . . . stay tuned.*

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's *dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		**−10±2%**
Comparisons	2	1.9	**−5%**
Swaps	$0.\overline{3}$	0.6	+80%
Bytecode Instructions	18	21.7	+20.6%
MMIX oops υ	11	13.1	+19.1%
MMIX mems μ	$2.\overline{6}$	2.8	+5%
scanned elements[1] *(≈ cache misses)*	2	1.6	**−20%**

$\cdot n \ln n + O(n)$, *average case results*

*What about **branch misses**? Can they **explain** YQS's success? . . . stay tuned.*

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Track Record of Dual-Pivot Quicksort

- Since 2009, Java uses **YAROSLAVSKIY's *dual-pivot Quicksort (YQS)***
- **faster** than previously used *classic Quicksort (CQS)* in practice
- **traditional** cost measures do **not** explain this!

	CQS	YQS	Relative
Running Time	(from various experiments)		**−10±2%**
Comparisons	2	1.9	**−5%**
Swaps	$0.\overline{3}$	0.6	+80%
Bytecode Instructions	18	21.7	+20.6%
MMIX oops υ	11	13.1	+19.1%
MMIX mems μ	$2.\overline{6}$	2.8	+5%
scanned elements[1] *(≈ cache misses)*	2	1.6	**−20%**

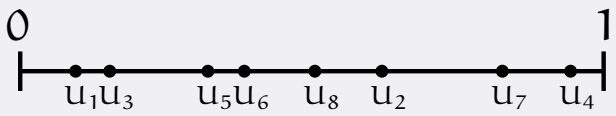
$\cdot n \ln n + O(n)$, *average case results*

*What about **branch misses**? Can they **explain** YQS's success? ... stay tuned.*

[1] KUSHAGRA, LÓPEZ-ORTIZ, MUNRO, QIAO; *ALENEX 2014*

Random Model

- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



- pairwise **distinct** almost surely
- relative ranking is a **random permutation**

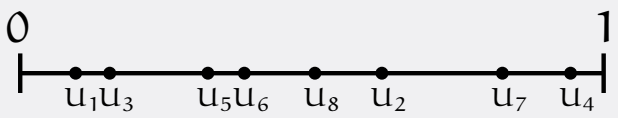
$\rightsquigarrow$ equivalent to classic model

- Consider **pivot value P fixed**:
  - $\Pr[\ U < P\ ] = P$
  - $\Pr[\ U > P\ ] = 1 - P$

- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\ U < P\ ] = D_1$
  - $\Pr[\ P < U < Q\ ] = D_2$
  - $\Pr[\ U > Q\ ] = D_3$

Random Model

- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



  - pairwise **distinct** almost surely
  - relative ranking is a **random permutation**

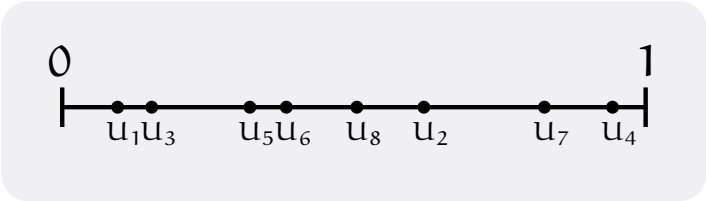
$\rightsquigarrow$ equivalent to classic model

- Consider **pivot value P fixed**:
  - $\Pr[\ U < P\ ] = P$
  - $\Pr[\ \quad\quad\ ] = \quad$

- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\ U < P\ ] = D_1$
  - $\Pr[\ P < U < Q\ ] = D_2$
  - $\Pr[\ U > Q\ ] = D_3$

Random Model

- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



- pairwise **distinct** almost surely
- relative ranking is a **random permutation**

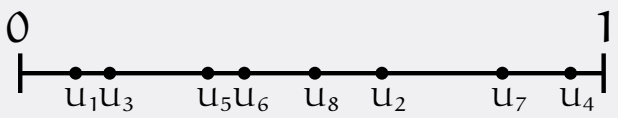
$\rightsquigarrow$ equivalent to classic model

- Consider **pivot value P fixed**:
  - $\Pr[\ U < P\ ] = \boxed{\ P\ }$
  - $\Pr[\ \boxed{\ P < U < Q\ }\ ] = \boxed{\ D_2\ }$
  - $\Pr[\ \boxed{\ U > Q\ }\ ] = \boxed{\ D_3\ }$

- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\ \boxed{\ U < P\ }\ ] = \boxed{\ D_1\ }$
  - $\Pr[\ \boxed{\ P < U < Q\ }\ ] = \boxed{\ D_2\ }$
  - $\Pr[\ \boxed{\ U > Q\ }\ ] = \boxed{\ D_3\ }$

Random Model

- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



  - pairwise **distinct** almost surely
  - relative ranking is a **random permutation**

⇝ equivalent to classic model

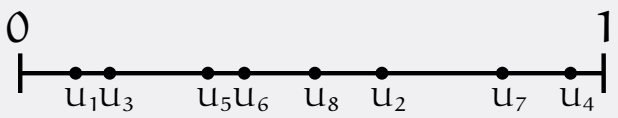
- Consider **pivot value P fixed**:
  - $\Pr[\, u < P \,] = P$
  - $\Pr[\, u > P \,] = 1 - P$



- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\, u < P \,] = D_1$
  - $\Pr[\, P < u < Q \,] = D_2$
  - $\Pr[\, u > Q \,] = D_3$

Random Model

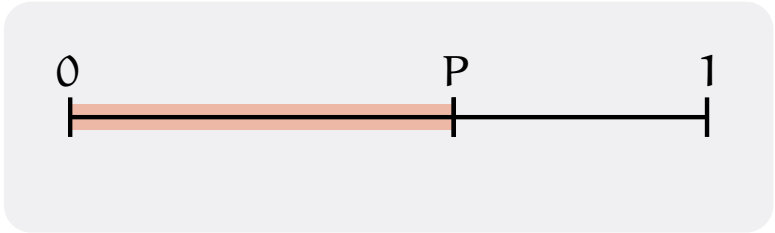
- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



  - pairwise **distinct** almost surely
  - relative ranking is a **random permutation**

⇝ equivalent to classic model

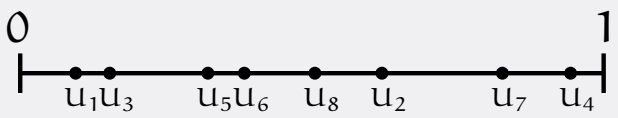
- Consider **pivot value P fixed**:
  - $\Pr[\, U < P \,] = P$
  - $\Pr[\, U > P \,] = 1 - P$



- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\, U < P \,] = D_1$
  - $\Pr[\, P < U < Q \,] = D_2$
  - $\Pr[\, U > Q \,] = D_3$

Random Model

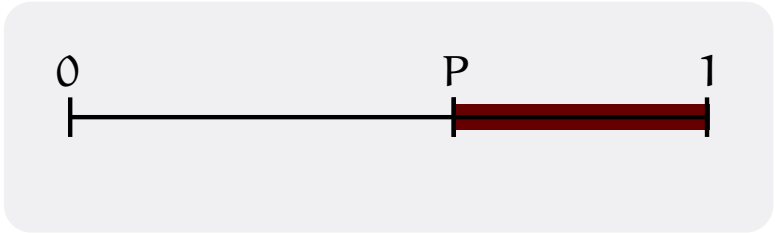
- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



  - pairwise **distinct** almost surely
  - relative ranking is a **random permutation**

⇝ equivalent to classic model

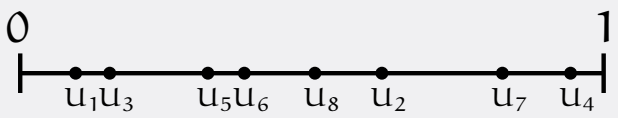
- Consider **pivot value P fixed**:
  - $\Pr[\, u < P \,] = P$
  - $\Pr[\, u > P \,] = 1 - P$



- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\, u < P \,] = D_1$
  - $\Pr[\, P < u < Q \,] = D_2$
  - $\Pr[\, u > Q \,] = D_3$

Random Model

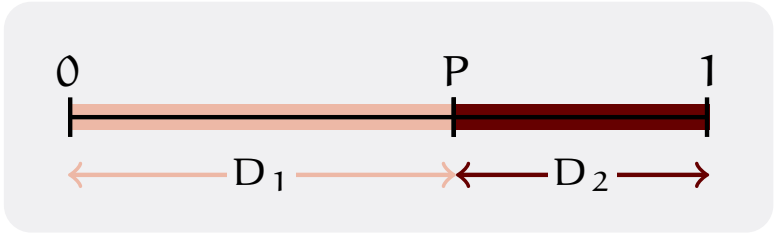
- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



  - pairwise **distinct** almost surely
  - relative ranking is a **random permutation**

⇝ equivalent to classic model

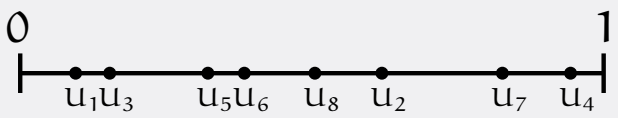
- Consider **pivot value P fixed**:
  - $\Pr[\, u < P \,] = P = D_1$
  - $\Pr[\, u > P \,] = 1 - P = D_2$



- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\, u < P \,] = D_1$
  - $\Pr[\, P < u < Q \,] = D_2$
  - $\Pr[\, u > Q \,] = D_3$

Random Model

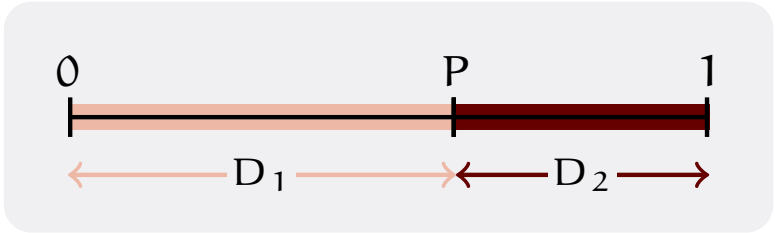
- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$



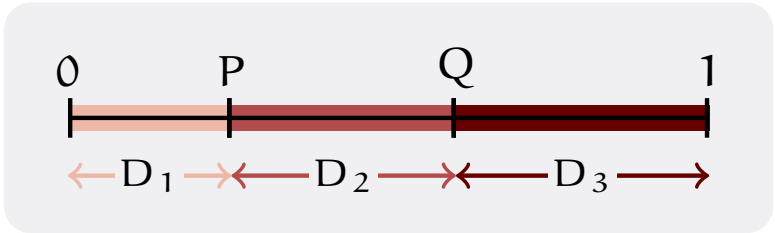
  - pairwise **distinct** almost surely
  - relative ranking is a **random permutation**

⇝ equivalent to classic model

- Consider **pivot value P fixed**:
  - $\Pr[\, U < P \,] = P = D_1$
  - $\Pr[\, U > P \,] = 1 - P = D_2$



- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\, U < P \,] = D_1$
  - $\Pr[\, P < U < Q \,] = D_2$
  - $\Pr[\, U > Q \,] = D_3$



Random Model

- n **i. i. d.** elements chosen **uniformly** in $[0, 1]$

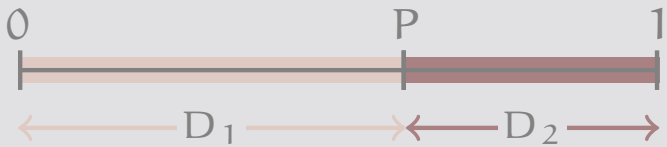


- pairwise **distinct** almost surely

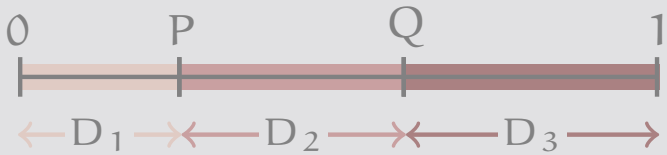
on

These probabilities hold for all elements U, **independent** of all other elements!

- Consider **pivot value P fixed**:
  - $\Pr[\,U < P\,] = P = D_1$
  - $\Pr[\,U > P\,] = 1 - P = D_2$



- Similarly for **dual-pivot** Quicksort with pivots $P \leqslant Q$
  - $\Pr[\,U < P\,] = D_1$
  - $\Pr[\,P < U < Q\,] = D_2$
  - $\Pr[\,U > Q\,] = D_3$



Branches in CQS

How many branches *in first partitioning step of CQS?*

- one **comparison branch** per element U:
  - $U[i] < P$ $\rightsquigarrow$ left partition
  - $U[i] \geq P$ $\rightsquigarrow$ right partition
- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

Branches in CQS

How many branches *in first partitioning step of CQS?*

- one **comparison branch** per element $\mathfrak{U}$:
  - $\mathfrak{U} < \mathfrak{P}$ $\rightsquigarrow$ left partition
  - $\mathfrak{U} > \mathfrak{P}$ $\rightsquigarrow$ right partition

- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

  $\rightsquigarrow$ can be **ignored** (for leading term asymptotics)

Branches in CQS

How many branches *in first partitioning step of CQS?*

- one **comparison branch** per element U:
  - $U < P$ $\rightsquigarrow$ left partition
  - $U > P$ $\rightsquigarrow$ right partition
- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

  $\rightsquigarrow$ can be **ignored** (for leading term asymptotics)

Branches in CQS

How many branches *in first partitioning step of CQS?*

- one **comparison branch** per element $\mathtt{U}$:
  - $\mathtt{U} < \mathtt{P}$ $\rightsquigarrow$ left partition
  - $\mathtt{U} > \mathtt{P}$ $\rightsquigarrow$ right partition

- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

  $\rightsquigarrow$ can be **ignored** (for leading term asymptotics)

Branches in CQS

How many branches *in first partitioning step of CQS?*

Consider **pivot** value P **fixed**. $\rightsquigarrow \mathbf{D} = (D_1, D_2) = (P, 1-P)$ fixed.

- one **comparison branch** per element U:
  - $U < P$ $\rightsquigarrow$ left partition
  - $U > P$ $\rightsquigarrow$ right partition

- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

  $\rightsquigarrow$ can be **ignored** (for leading term asymptotics)

Branches in CQS

How many branches *in first partitioning step of CQS?*

Consider **pivot** value P **fixed**. $\rightsquigarrow \mathbf{D} = (D_1, D_2) = (P, 1-P)$ fixed.

- one **comparison branch** per element U:
  - $U < P$ $\rightsquigarrow$ left partition
  - $U > P$ $\rightsquigarrow$ right partition

  $\rightsquigarrow$ **branch taken** with **prob.** P
  $\rightsquigarrow$ **i. i. d.** for all elements U!
  $\rightsquigarrow$ ***memoryless*** source

- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

  $\rightsquigarrow$ can be **ignored** (for leading term asymptotics)

Branches in CQS

How many branches *in first partitioning step of CQS?*

Consider **pivot** value P **fixed**. $\rightsquigarrow \mathbf{D} = (D_1, D_2) = (P, 1-P)$ fixed.

- one **comparison branch** per element U:
  - $U < P$ $\rightsquigarrow$ left partition
  - $U > P$ $\rightsquigarrow$ right partition

  $\rightsquigarrow$ **branch taken** with **prob.** P **i. i. d.** for all elements U!
  $\rightsquigarrow$ ***memoryless*** *source*

- other branches (loop logic etc.)
  - easy to predict
  - only constant number of mispredictions

  $\rightsquigarrow$ can be **ignored** (for leading term asymptotics)

Misprediction Rate for Memoryless Sources

Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**

  $$\frac{H(p)}{\log(\text{\#Branches})}$$

  $$H(p) = -p\log(p) - (1-p)\log(1-p)$$

- ⇝ **But: Actual predictors have very little memory!**

- **1-bit Predictor**

  - Always predict the same as last time
  - Counter-intuitive
  - Simple

Misprediction Rate for Memoryless Sources

Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{OPT}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ **taken** | $\hat{p} < \frac{1}{2} \rightsquigarrow$ ***not*** **taken**

$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- **1-bit Predictor**
  - $\rightsquigarrow$ Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{1\text{-bit}}(p) = 2p(1-p)$

Misprediction Rate for Memoryless Sources

Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{\text{OPT}}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ taken $\hat{p} < \frac{1}{2} \rightsquigarrow$ *not* taken

$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- **1-bit Predictor**
  - $\rightsquigarrow$ Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{\text{1bit}}(p) = 2p(1-p)$

Misprediction Rate for Memoryless Sources

Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{OPT}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ **taken**	$\hat{p} < \frac{1}{2} \rightsquigarrow$ ***not*** **taken**

$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- **1-bit Predictor**
  - Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{1bit}(p) = 2p(1-p)$

Misprediction Rate for Memoryless Sources

Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{OPT}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ **taken**	$\hat{p} < \frac{1}{2} \rightsquigarrow$ ***not*** **taken**

$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- 1-bit Predictor
  - Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{1\text{-bit}}(p) = 2p(1-p)$

Misprediction Rate for Memoryless Sources

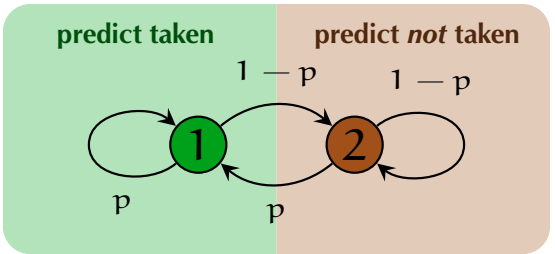
Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{OPT}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ **taken**	$\hat{p} < \frac{1}{2} \rightsquigarrow$ ***not*** **taken**

$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- **1-bit Predictor**
  - Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{1bit}(p) = 2p(1-p)$



Misprediction Rate for Memoryless Sources

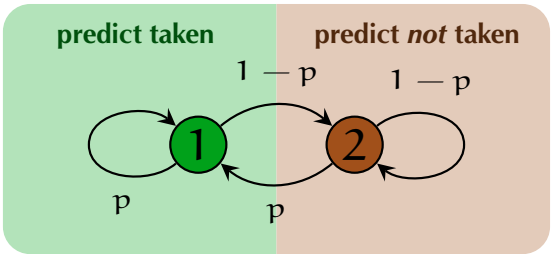
Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{OPT}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ **taken**	$\hat{p} < \frac{1}{2} \rightsquigarrow$ ***not*** **taken**

$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- **1-bit Predictor**
  - Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{1bit}(p) = 2p(1-p)$



Misprediction Rate for Memoryless Sources

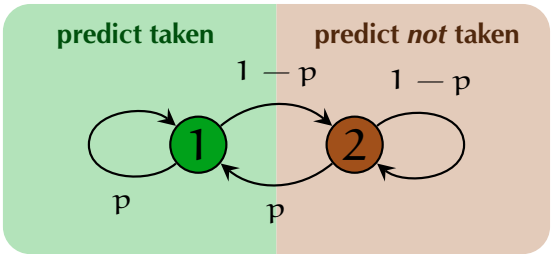
Branches taken i. i. d. with probability p.

- **Information theoretic lower bound:**
  - $\rightsquigarrow$ Miss rate: $f_{OPT}(p) = \min\{p, 1-p\}$
    - Can approach lower bound by estimating p.

$\hat{p} \geq \frac{1}{2} \rightsquigarrow$ **taken**	$\hat{p} < \frac{1}{2} \rightsquigarrow$ ***not*** **taken**

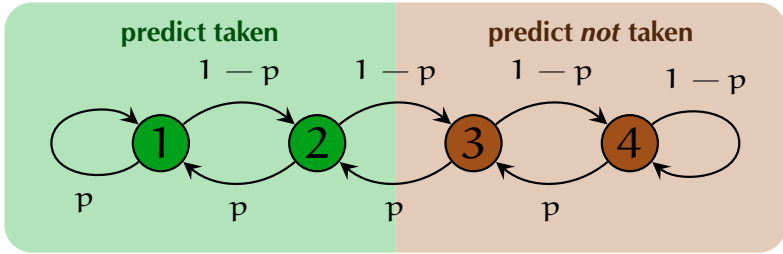
$\rightsquigarrow$ **But:** Actual predictors have very **little memory**!

- **1-bit Predictor**
  - Wrong prediction whenever value changes
  - $\rightsquigarrow$ Miss rate: $f_{1bit}(p) = 2p(1-p)$



Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? ...depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$

⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

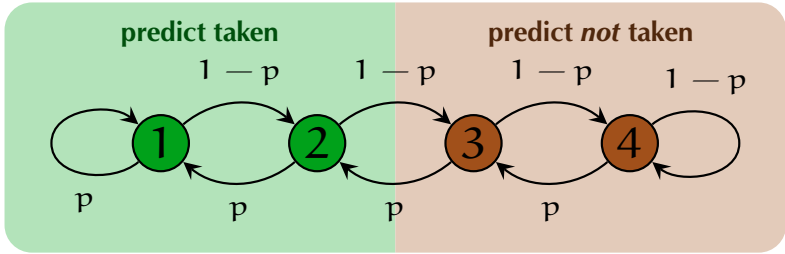
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$

⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

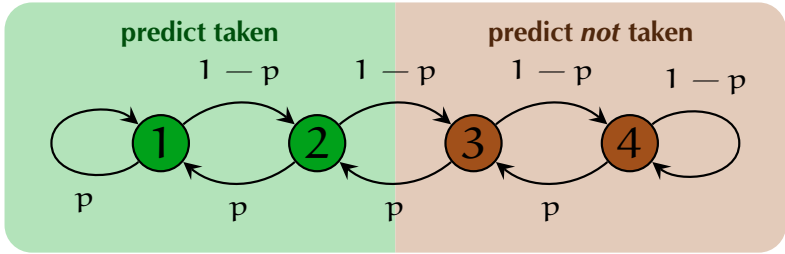
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$

$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

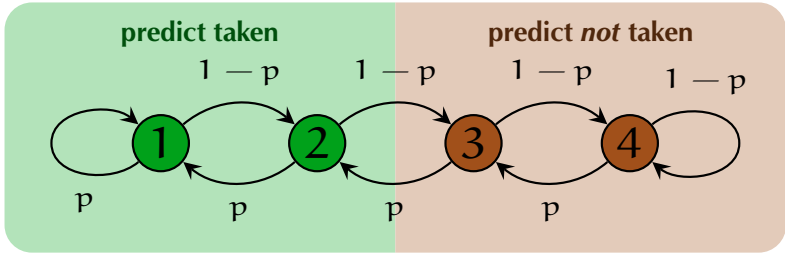
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

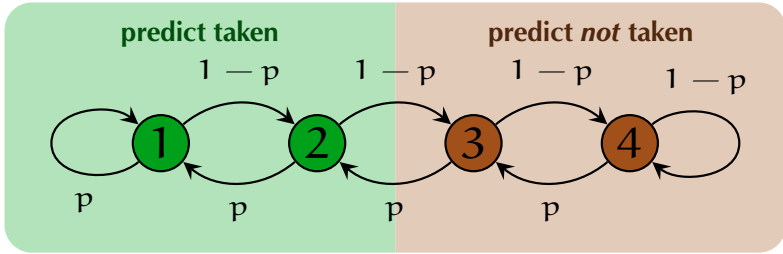
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

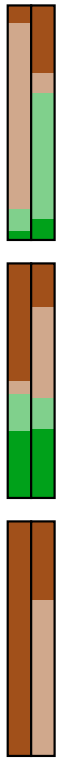
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? ... depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

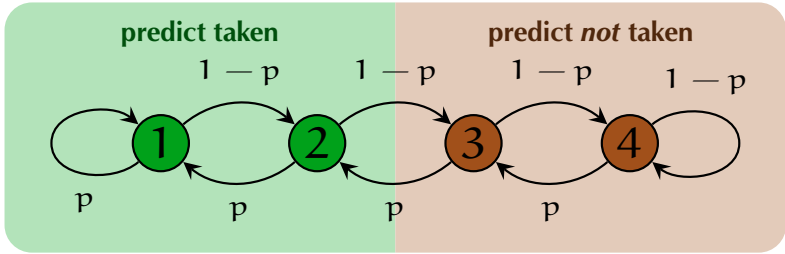
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

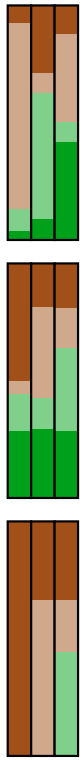
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

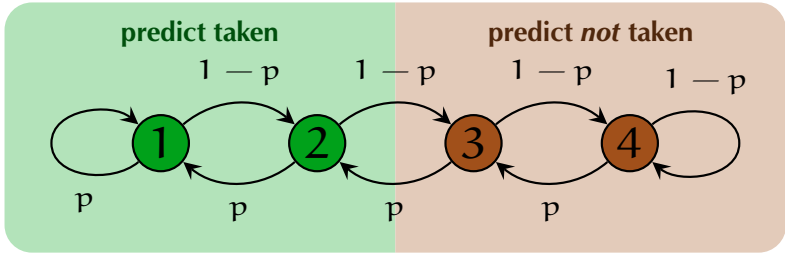
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

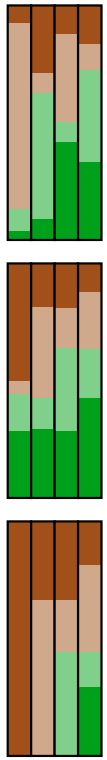
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

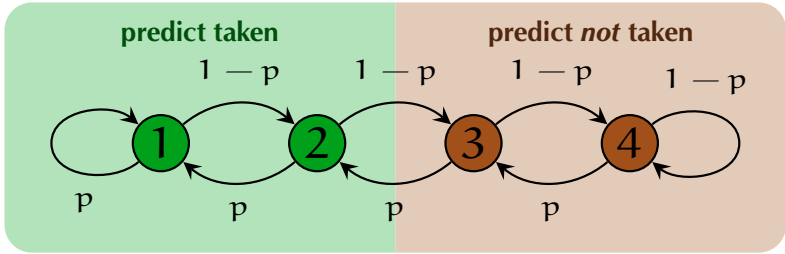
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

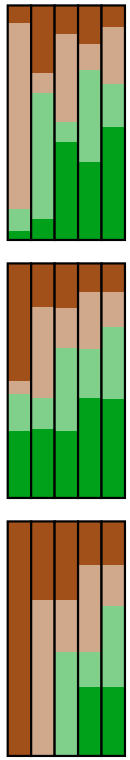
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

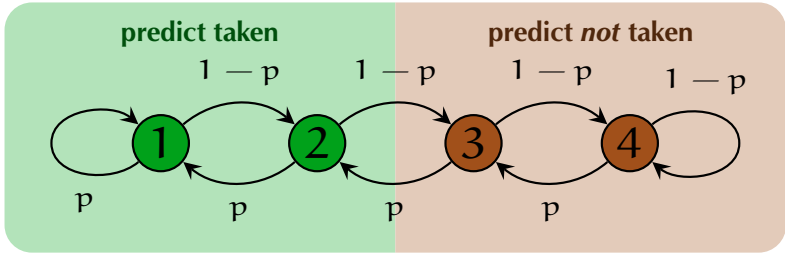
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

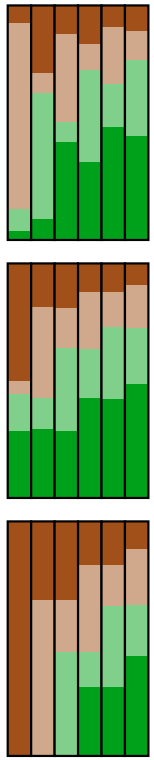
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

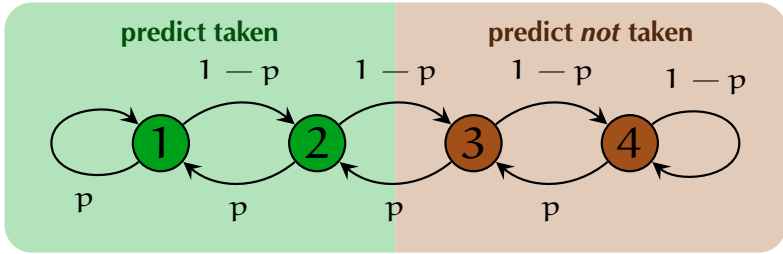
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

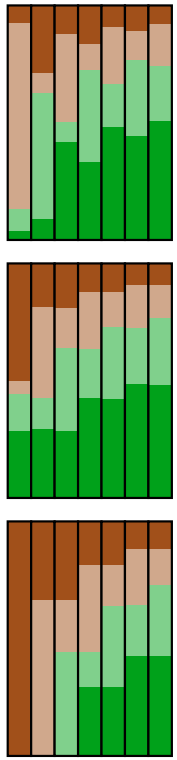
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? ... depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

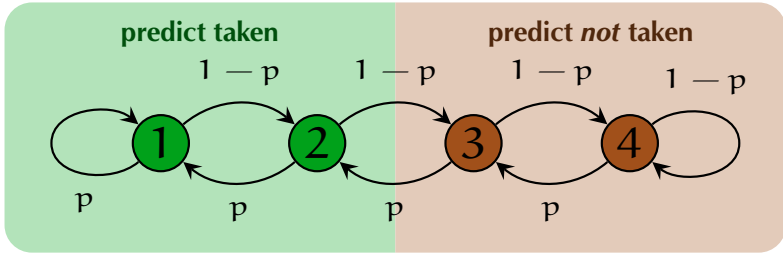
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

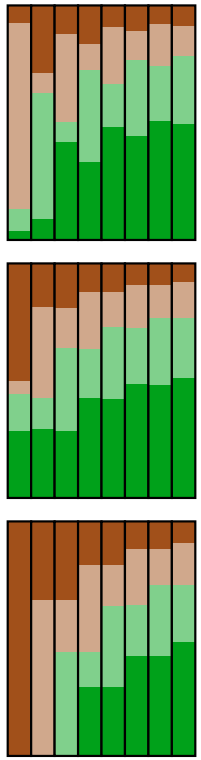
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? ... depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

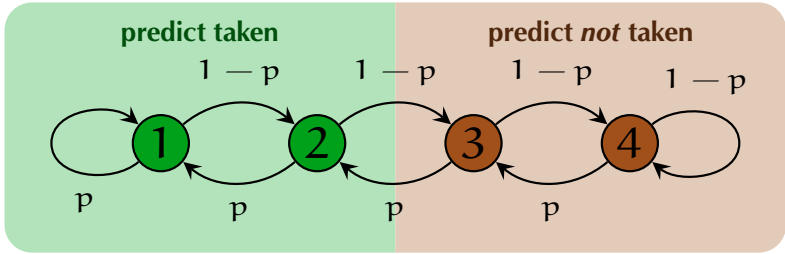
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

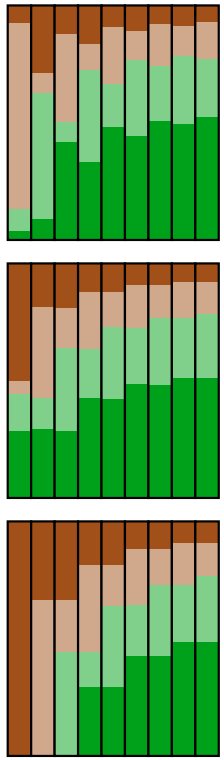
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

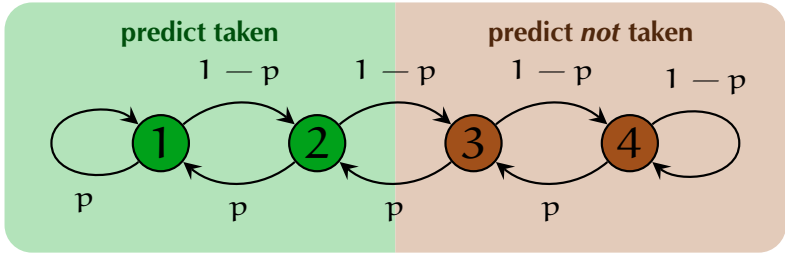
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

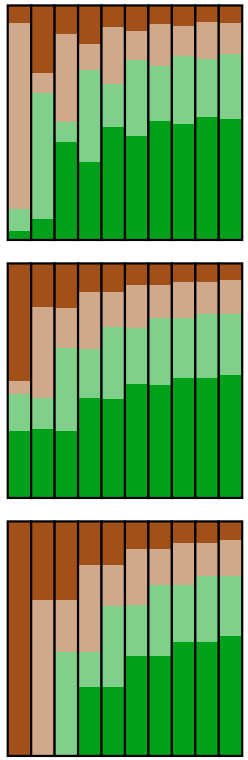
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

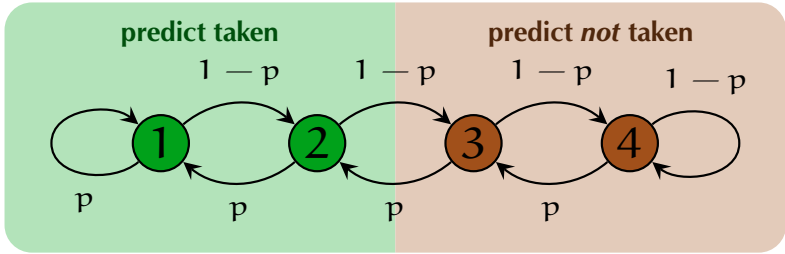
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

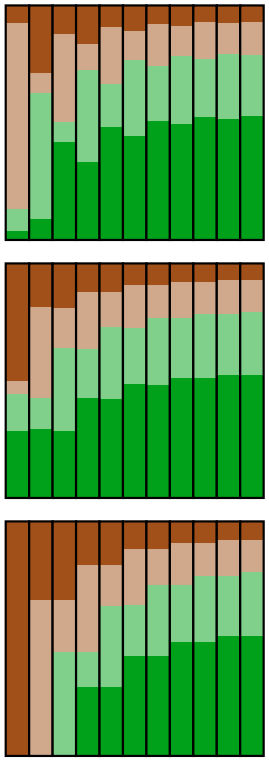
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

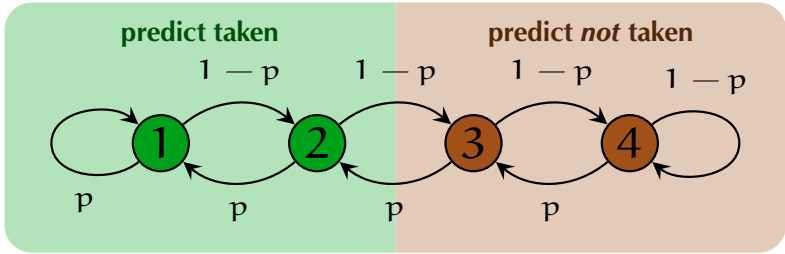
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

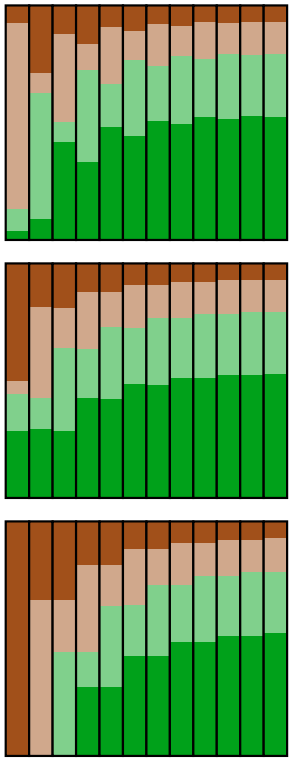
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

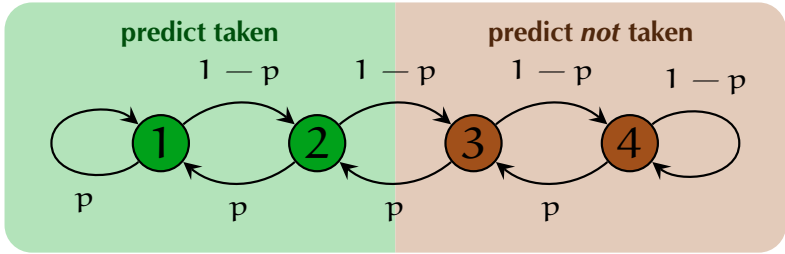
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

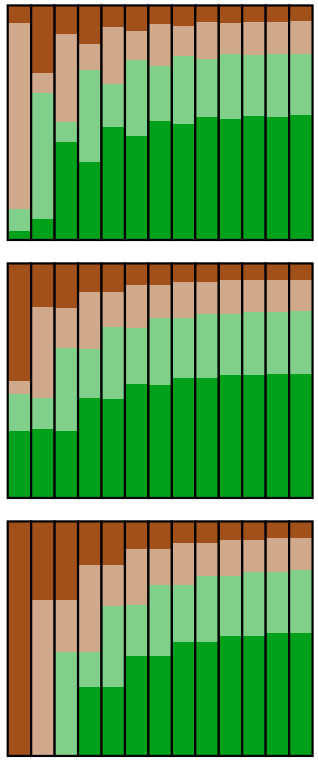
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

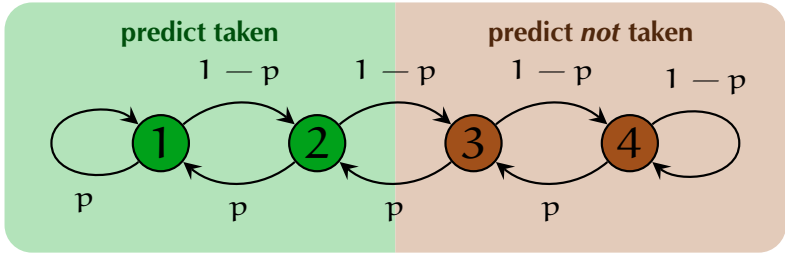
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

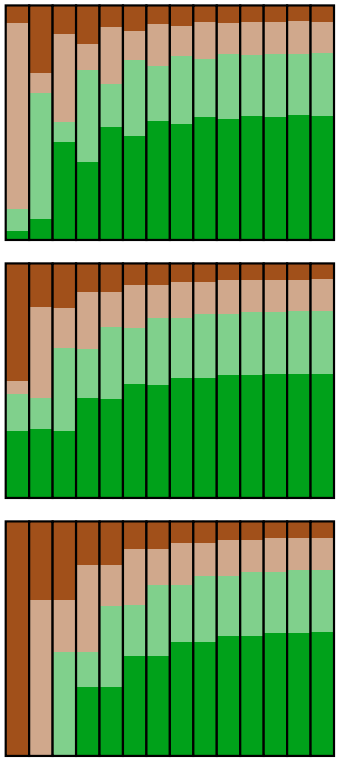
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

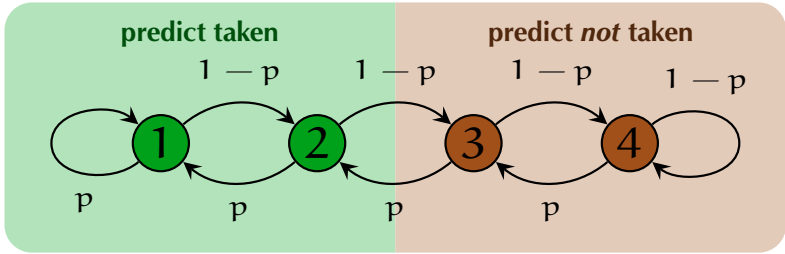
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

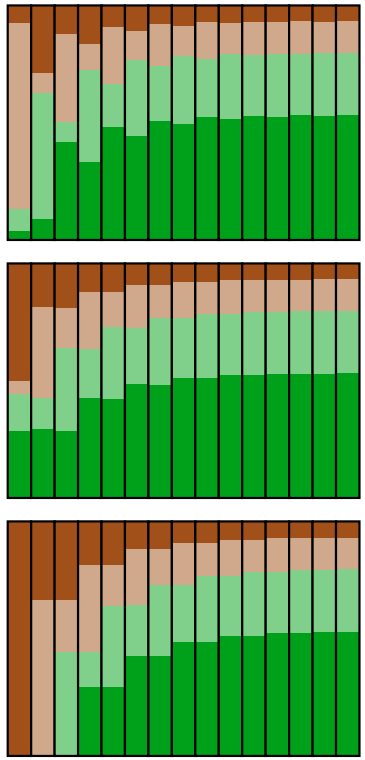
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

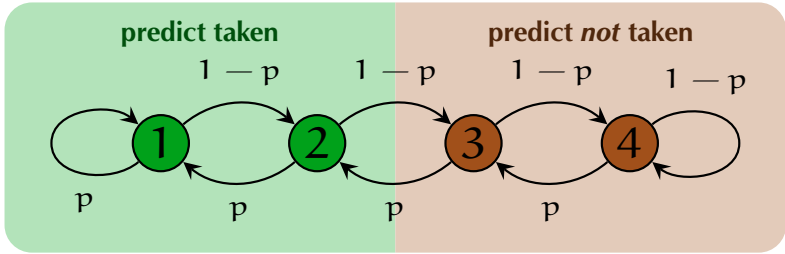
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

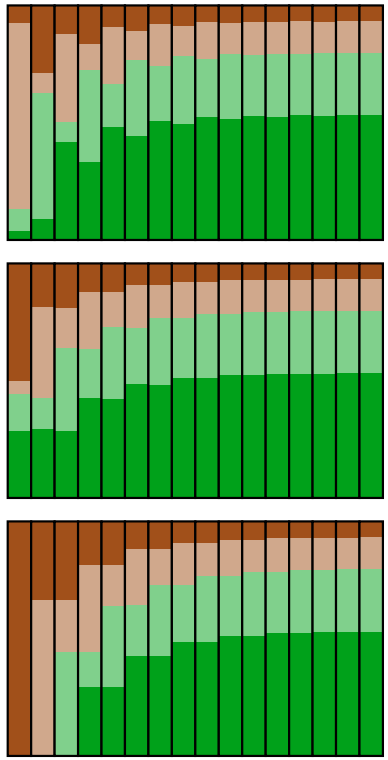
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

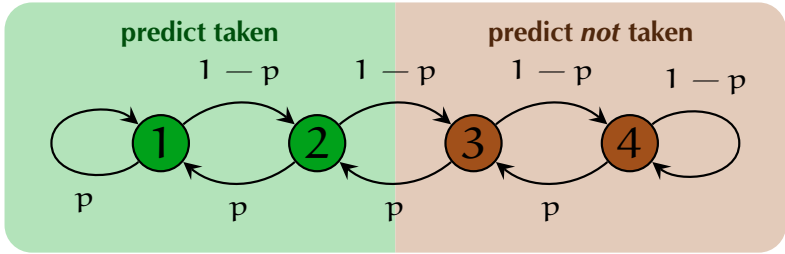
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

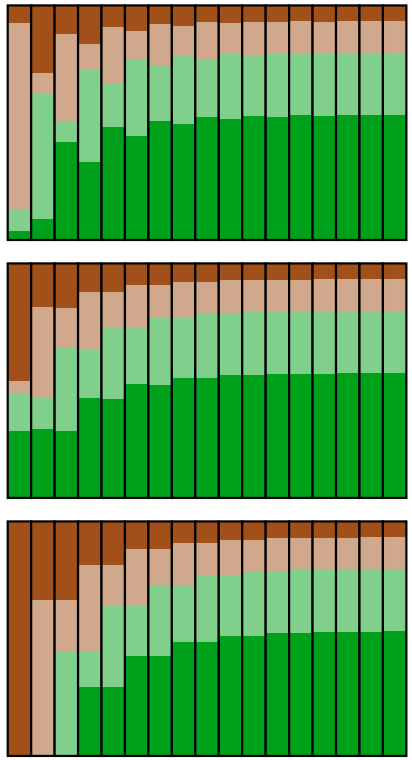
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

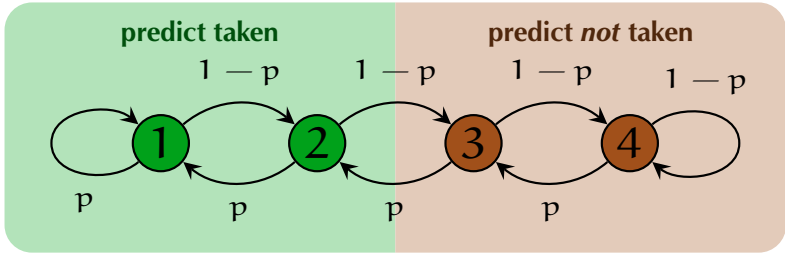
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

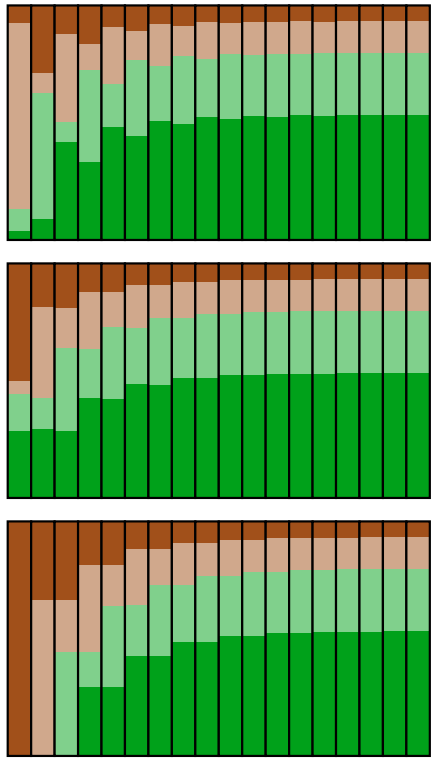
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? ... depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



$\rightsquigarrow$ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

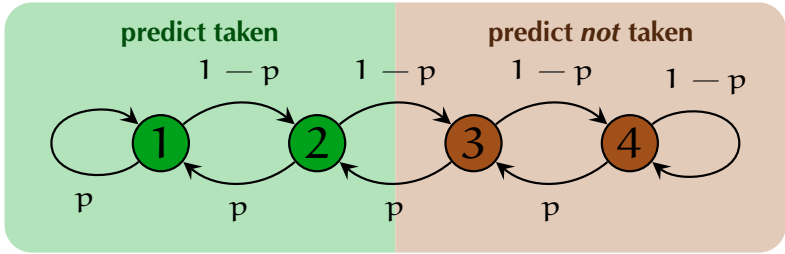
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

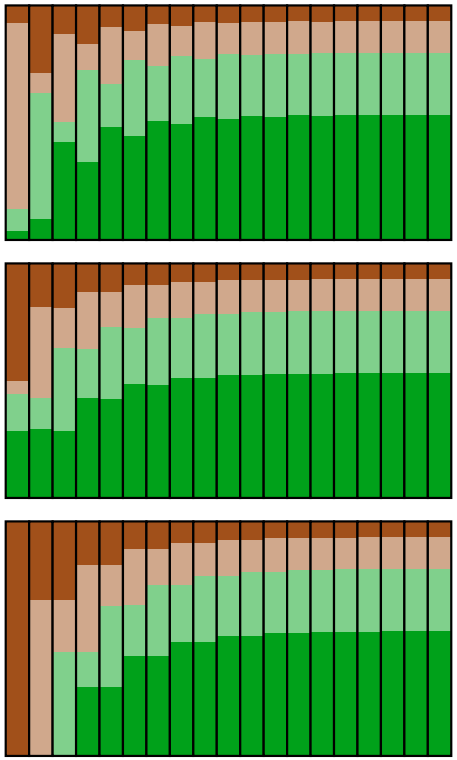
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

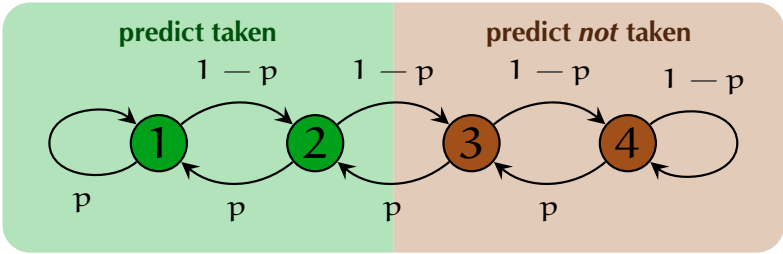
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

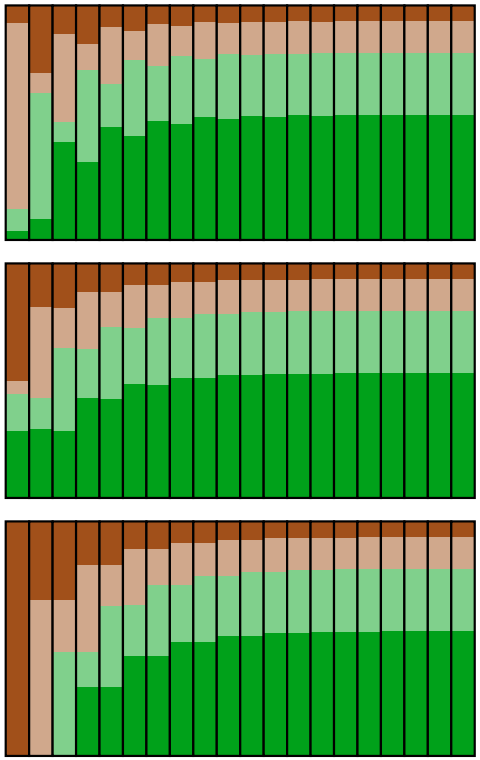
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

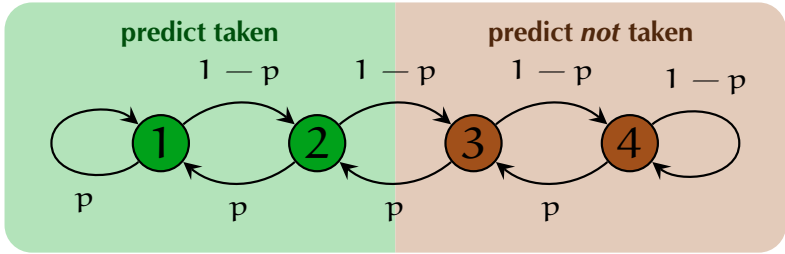
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

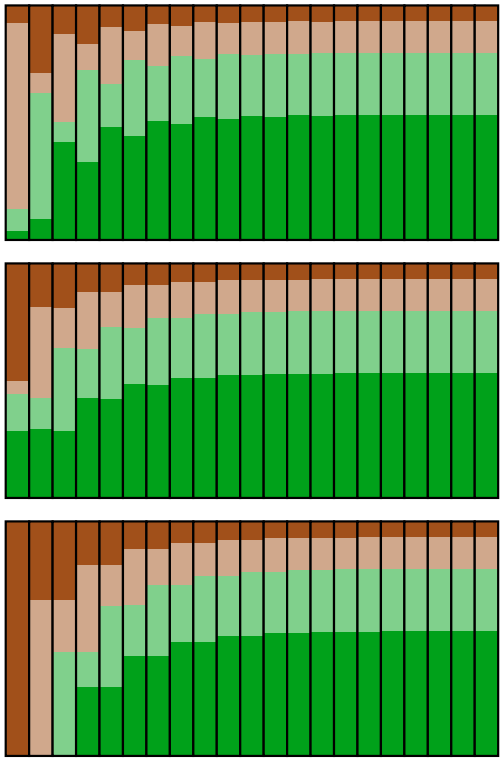
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

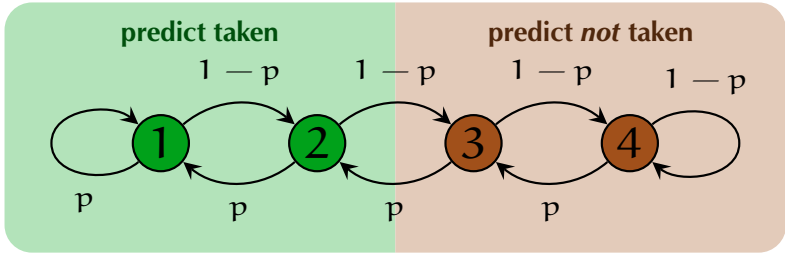
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

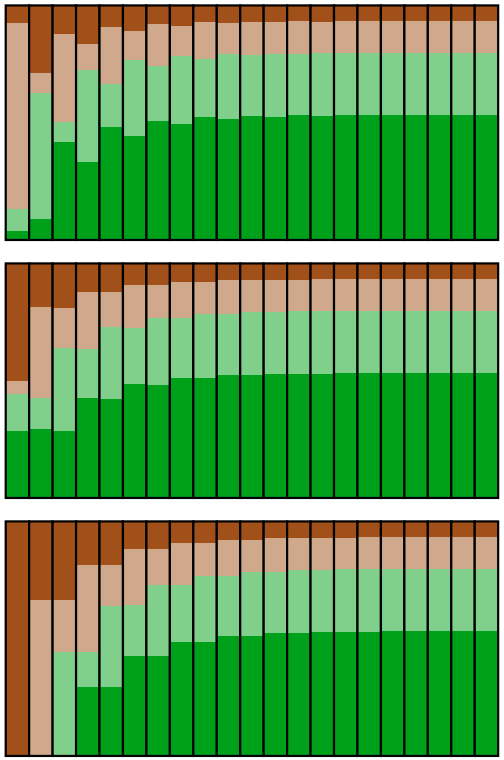
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

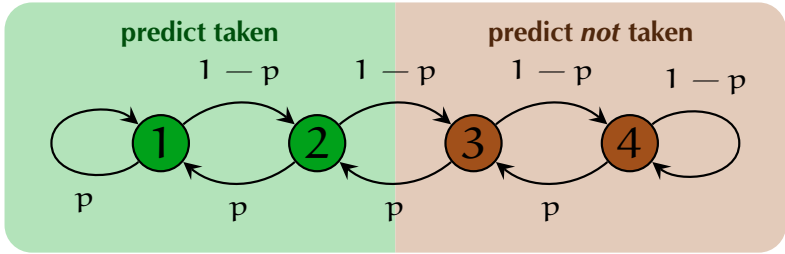
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

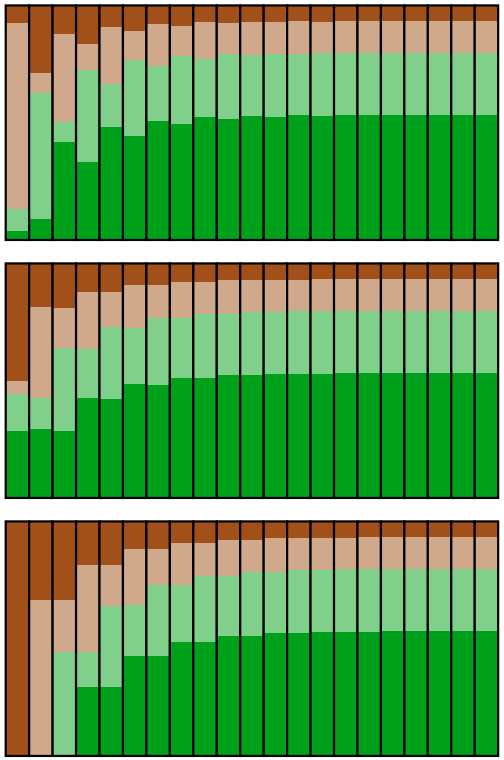
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? . . . depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

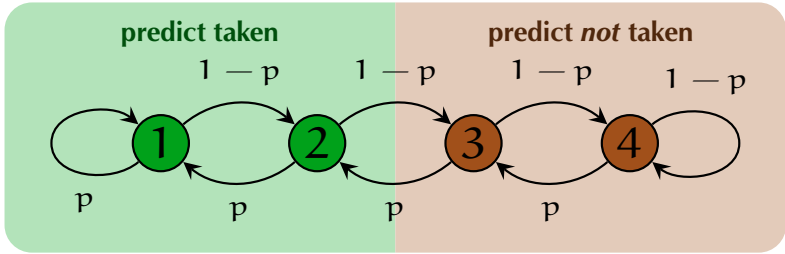
- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

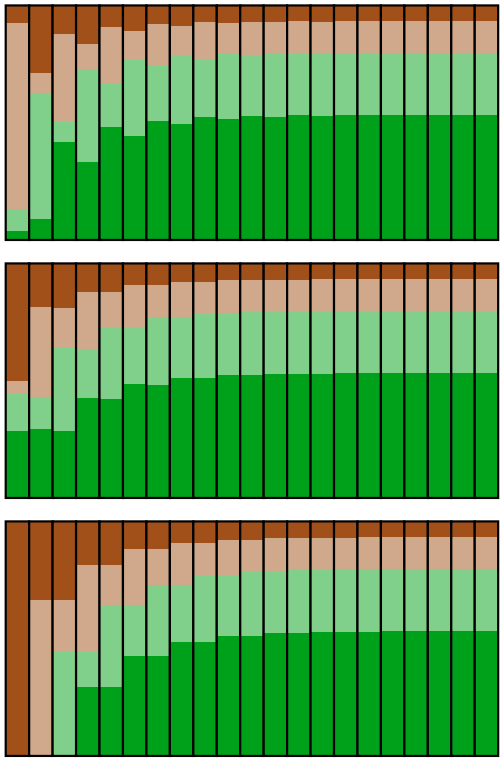
$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Misprediction Rate for Memoryless Sources [2]

- **2-bit Saturating Counter**
  - Miss rate? ... depends on state!



- **But:** Very fast convergence to **steady state**
  - different initial state distributions
  - 20 iterations for $p = \frac{2}{3}$



⇝ use ***steady-state miss-rate***:
expected miss rate over states in stationary distribution

- here: $f_{\text{2-bit-sc}}(p) = \dfrac{q}{1-2q}$ with $q = p(1-p)$.

- similarly for **2-bit Flip-Consecutive**

$$f_{\text{2-bit-fc}}(p) = \frac{q(1+2q)}{1-q}.$$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P?

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P?

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P?

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P?

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P*?*

- **without sampling:** $P \stackrel{\mathcal{D}}{=} \mathit{Uniform}(0, 1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")

- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3, 2)$:

- $\mathbf{t} = (0, 0)$ $\rightsquigarrow$ no sampling
- $\mathbf{t} = (t_1, t_2)$ gives $\mathrm{median}(t_1 + 1, t_2 + 1)$
- $\mathbf{t} = (t_1, t_2)$ gives $\mathrm{median}(t_1 + 1, t_2 + 1)$

$\rightsquigarrow$ Distribution of pivot value: $\boxed{P \stackrel{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)}$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

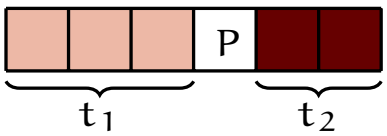
$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P*?*

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:



- $t = (0,0) \rightsquigarrow$ no sampling
- $t = (t,t)$ gives median-of-$(2t+1)$
- can also sample **skewed** pivots

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

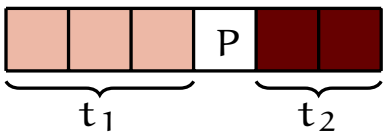
$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P?

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:



  - $\mathbf{t} = (0,0) \rightsquigarrow$ no sampling
  - $\mathbf{t} = (t,t)$ gives median-of-$(2t+1)$
  - can also sample **skewed** pivots

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

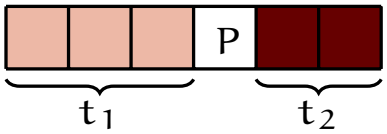
$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P*?*

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:



  - $\mathbf{t} = (0,0) \rightsquigarrow$ no sampling
  - $\mathbf{t} = (t,t)$ gives median-of-$(2t+1)$
  - can also sample **skewed** pivots

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

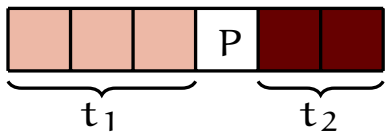
$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P*?*

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:



- $\mathbf{t} = (0,0) \rightsquigarrow$ no sampling
- $\mathbf{t} = (t,t)$ gives median-of-$(2t+1)$
- can also sample **skewed** pivots

$\rightsquigarrow$ Distribution of pivot value: $P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)$

Distribution of Pivot Values

- In (classic) Quicksort branch probability is **random**

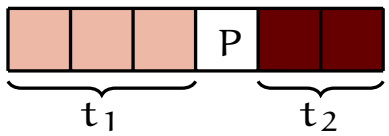
$\rightsquigarrow$ **expected** miss rate: $\mathbb{E}[f(P)]$. (expectation over pivot values P)

What is the distribution of P*?*

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
- **Typical pivot choice: median of k** (in practice: $k = 3$)

  or pseudomedian of 9 ("ninther")
- **Here:** more general scheme with parameter $\mathbf{t} = (t_1, t_2)$

  Example: $k = 6$ and $\mathbf{t} = (3,2)$:



  - $\mathbf{t} = (0,0) \rightsquigarrow$ no sampling
  - $\mathbf{t} = (t,t)$ gives median-of-$(2t+1)$
  - can also sample **skewed** pivots

$\rightsquigarrow$ Distribution of pivot value: $\boxed{P \overset{\mathcal{D}}{=} \mathit{Beta}(t_1 + 1, t_2 + 1)}$

Miss Rates for Quicksort Branch

- expected miss rate given by integral

$$\mathbb{E}[f(P)] = \int_0^1 f(p) \cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp$$

- e. g. for 1-bit predictor

$$\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p) \cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp$$

- **no concise** representation for other integrals . . . *(see paper)*

but: exact values for fixed t

Miss Rates for Quicksort Branch

- expected miss rate given by integral

$$\mathbb{E}[f(P)] = \int_0^1 f(p) \cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp$$

- e. g. for 1-bit predictor

$$\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p) \cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp$$

- **no concise** representation for other integrals . . . *(see paper)*

  but: exact values for fixed t

Miss Rates for Quicksort Branch

- expected miss rate given by integral

$$\mathbb{E}[f(P)] \;=\; \int_0^1 f(p)\cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp$$

- e. g. for 1-bit predictor

$$\mathbb{E}[f_{\text{1-bit}}(P)] \;=\; \int_0^1 2p(1-p)\cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp \;=\; 2\frac{(t_1+1)(t_2+1)}{(k+2)(k+1)}$$

- **no concise** representation for other integrals . . . *(see paper)*

  but: exact values for fixed t

Miss Rates for Quicksort Branch

- expected miss rate given by integral

$$\mathbb{E}[f(P)] \;=\; \int_0^1 f(p) \cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp$$

- e. g. for 1-bit predictor

$$\mathbb{E}[f_{\text{1-bit}}(P)] \;=\; \int_0^1 2p(1-p) \cdot \frac{p^{t_1}(1-p)^{t_2}}{B(\mathbf{t}+1)}\,dp \;=\; 2\frac{(t_1+1)(t_2+1)}{(k+2)(k+1)}$$

- **no concise** representation for other integrals . . . *(see paper)*

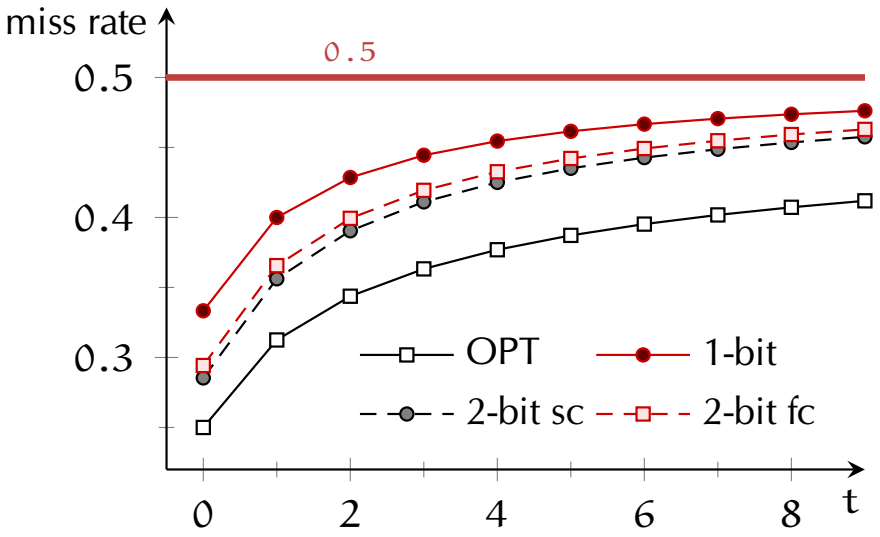
  but: exact values for fixed $\mathbf{t}$

Miss Rate and Branch Misses

Miss Rate for CQS with median of $2t+1$:

Miss Rate and Branch Misses

Miss Rate for CQS with median of $2t+1$:



- miss rates quickly get bad (close to guessing!)
- **but:** less **comparisons** in total!

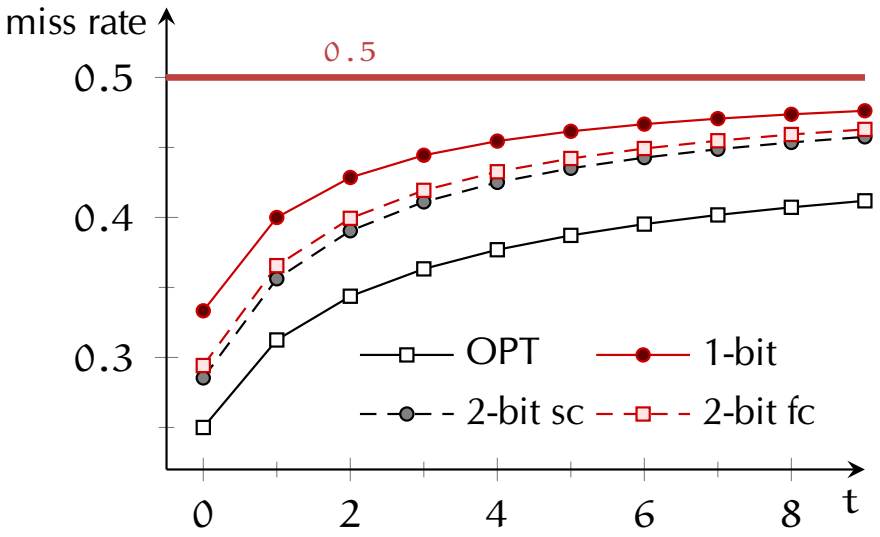
$\rightsquigarrow$ Consider **number of branch misses**:

$$\#\mathrm{BM} = \#\mathit{comparisons} \cdot \text{miss rate}$$

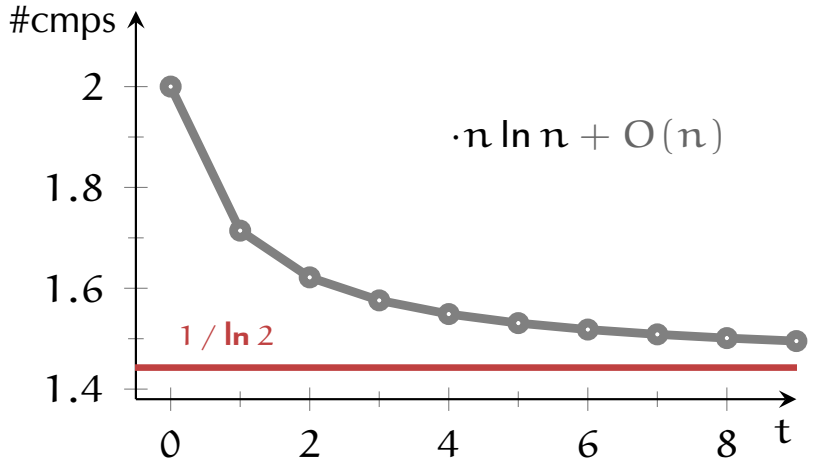
$\rightsquigarrow$ Overall BM still **grows** with t.

Miss Rate and Branch Misses

Miss Rate for CQS with median of $2t+1$:



- miss rates quickly get bad (close to guessing!)
- **but:** less **comparisons** in total!



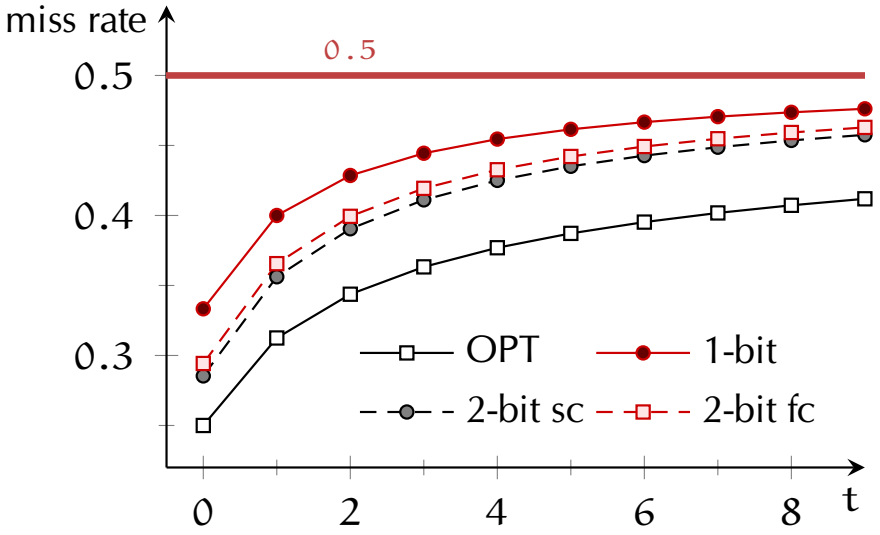
$\rightsquigarrow$ Consider **number of branch misses**:

$$\#\mathrm{BM} = \#\mathit{comparisons} \cdot \text{miss rate}$$

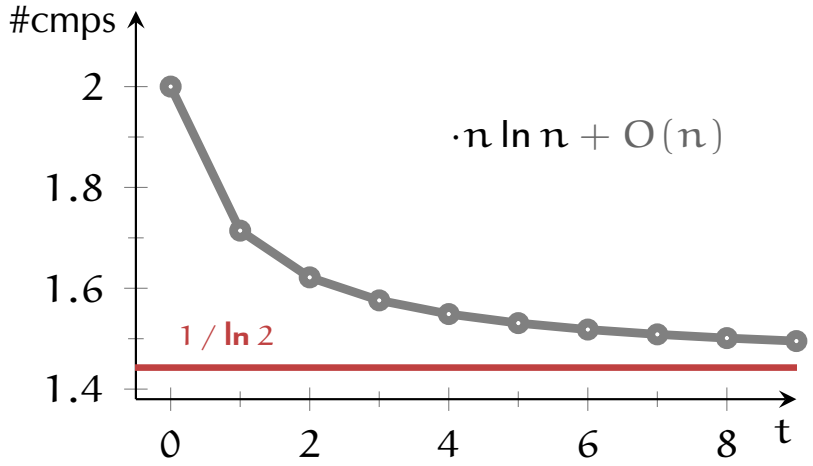
$\rightsquigarrow$ Overall BM still **grows** with t.

Miss Rate and Branch Misses

Miss Rate for CQS with median of $2t+1$:



- miss rates quickly get bad (close to guessing!)
- **but:** less **comparisons** in total!



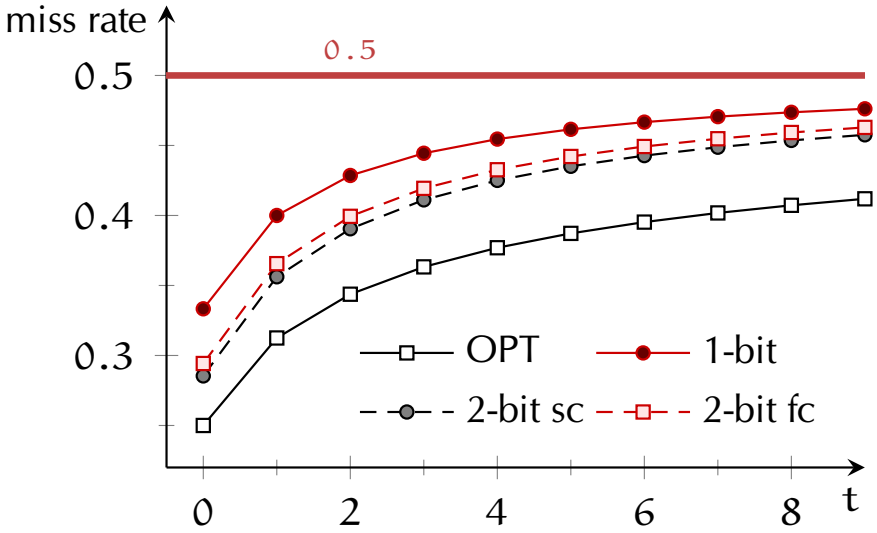
⇝ Consider **number of branch misses**:

$$\#\text{BM} = \#\textit{comparisons} \cdot \text{miss rate}$$

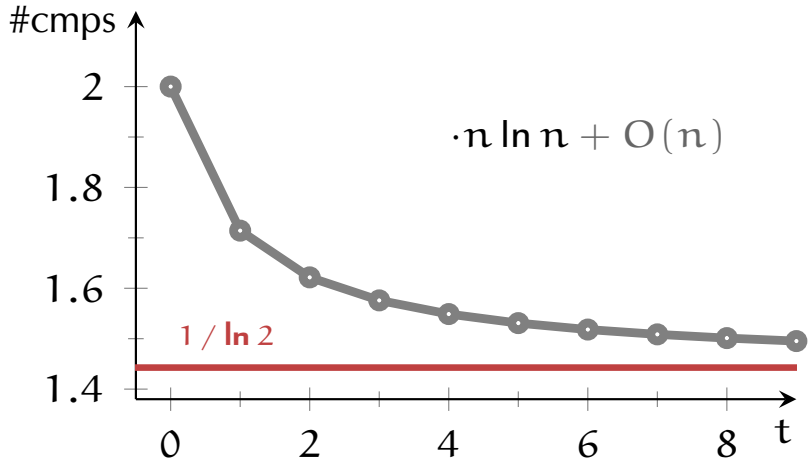
⇝ Overall BM still **grows** with t.

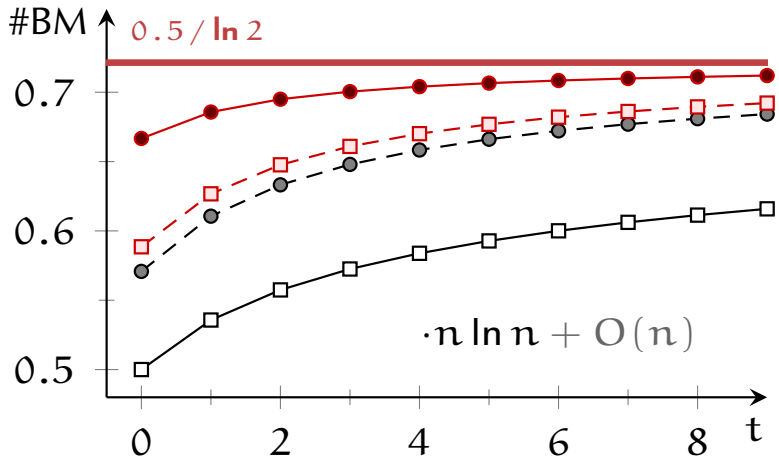
Miss Rate and Branch Misses

Miss Rate for CQS with median of $2t+1$:



- miss rates quickly get bad (close to guessing!)
- **but:** less **comparisons** in total!





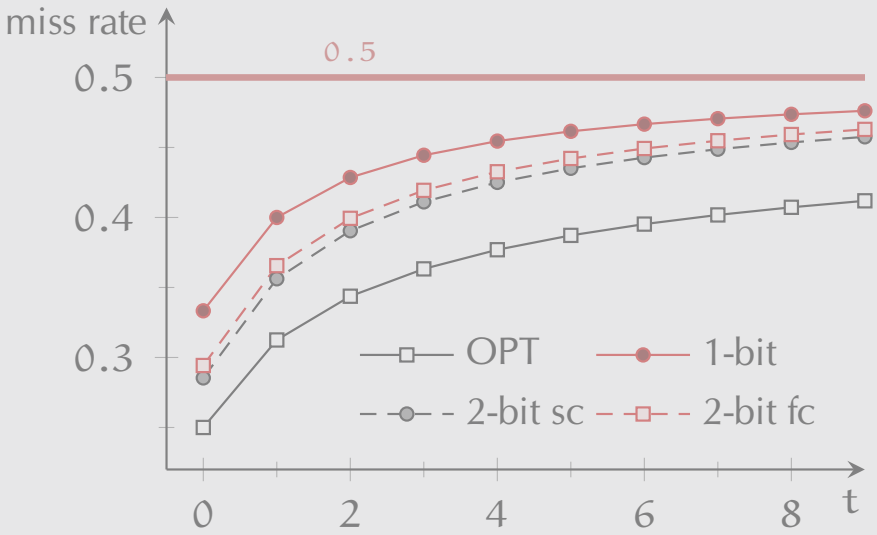
⇝ Consider **number of branch misses**:

$$\#\mathrm{BM} = \#\mathit{comparisons} \cdot \text{miss rate}$$

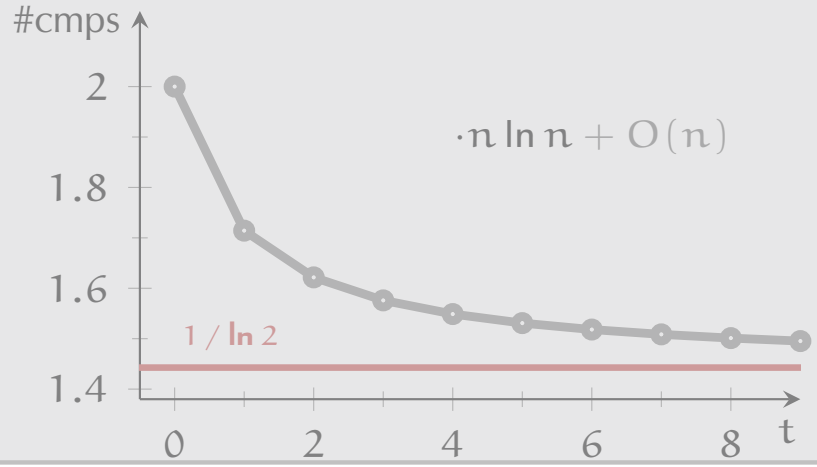
⇝ Overall BM still **grows** with t.

Miss Rate and Branch Misses

Miss Rate for CQS with median of $2t+1$:



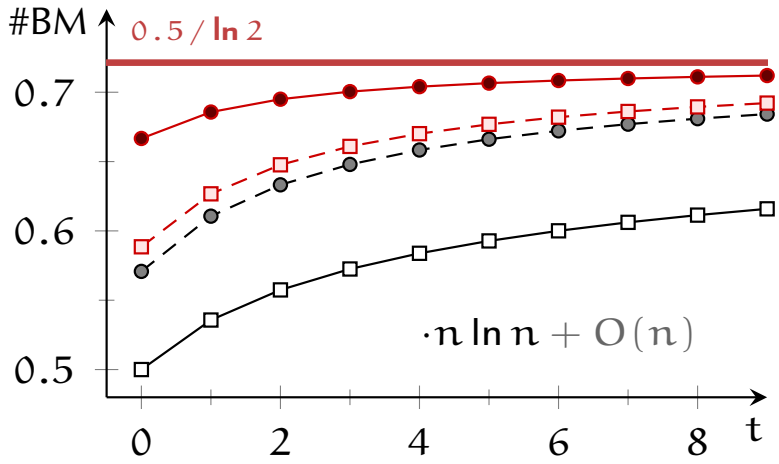
- miss rates quickly get bad (close to guessing!)
- **but:** less **comparisons** in total!



↝ Consider **number of branch misses**:

$$\#\mathrm{BM} = \#\mathit{comparisons} \cdot \text{miss rate}$$

↝ Overall BM still **grows** with t.



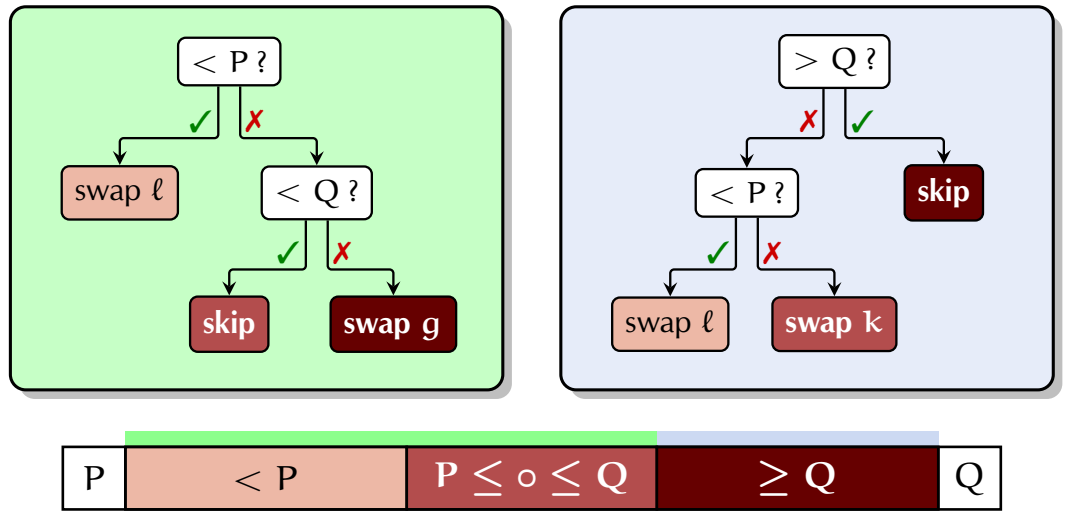
Branch Misses in YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

Branch Misses in YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

- Complication for analysis:
  - 4 branch locations
  - how often they are executed depends on input



- **Example:** $C^{(y1)}$
  - executed $(\mathrm{D}_1 + \mathrm{D}_2)n + \mathrm{O}(1)$ times. (in expectation, conditional on $\mathbf{D}$)
  - branch taken i i d with prob. D_1 (conditional on $\mathbf{D}$)
  - $\rightsquigarrow$ expected #BM at $\mathrm{C}^{(y1)}$ in first partitioning step:
    $\mathrm{D}_1(\mathrm{D}_1 + \mathrm{D}_2) \cdot \mathrm{I}(\mathrm{D}_1)\, n + \mathrm{O}(1)$
- Integrals even more "fun" … but doable

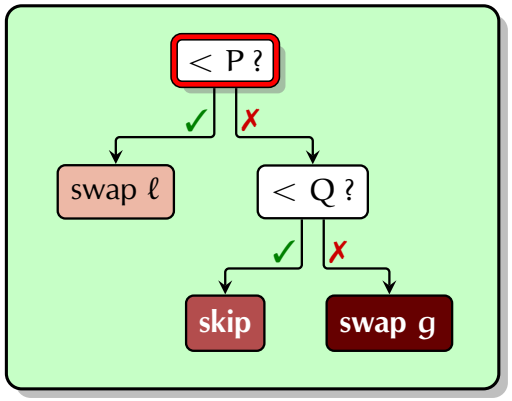
Branch Misses in YQS

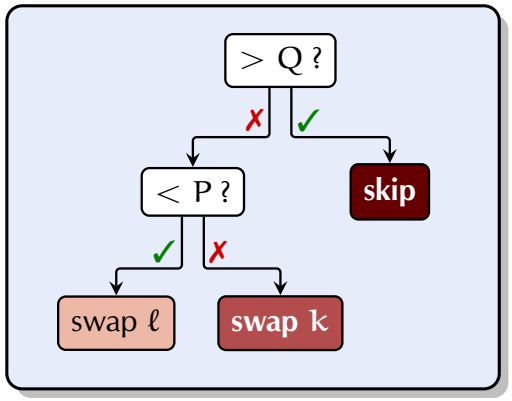
Original question: ***Does YQS better than CQS w. r. t. branch misses?***

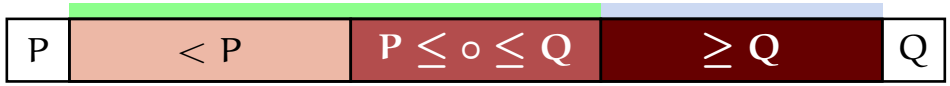
- Complication for analysis:
  - 4 branch locations
  - how often they are executed depends on input
- **Example:** $C^{(y1)}$
  - executed $(D_1 + D_2)n + O(1)$ times. (in expectation, conditional on $\mathbf{D}$)
  - branch taken i. i. d. with prob D_1. (conditional on $\mathbf{D}$)

  $\rightsquigarrow$ expected #BM at $C^{(y1)}$ in first partitioning step:

  $\mathbb{E}[(D_1 + D_2) \cdot f(D_1)] \cdot n + O(1)$
- Integrals even more "fun" ... but doable



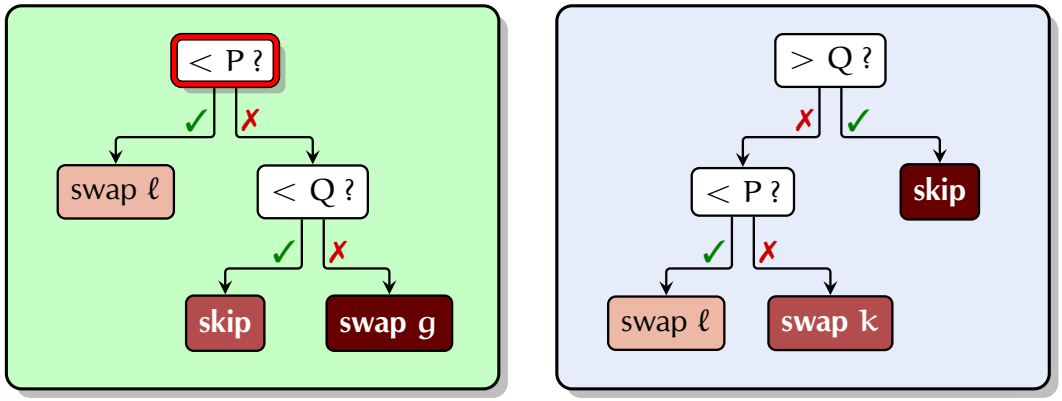




Branch Misses in YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

- Complication for analysis:
  - 4 branch locations
  - how often they are executed depends on input



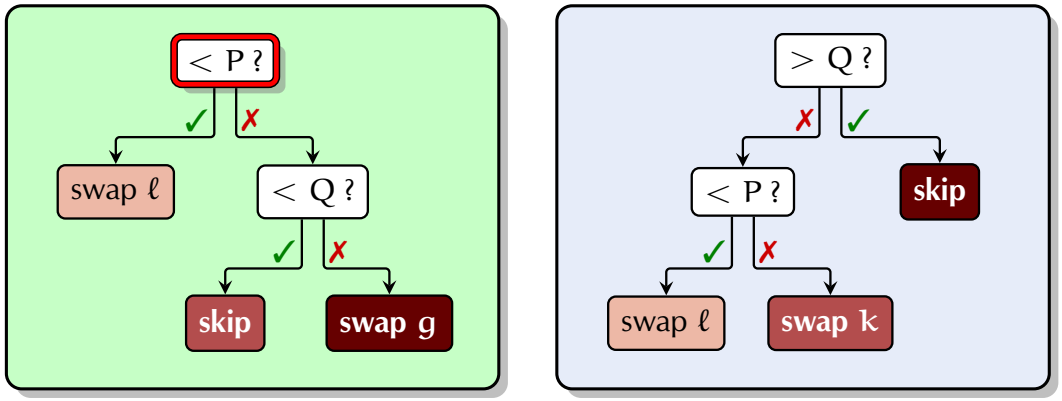
- **Example:** $C^{(y1)}$
  - executed $(\ D_1 + D_2\ )n + O(1)$ times. (in expectation, conditional on $\mathbf{D}$)
  - branch taken i. i. d. with prob D_1. (conditional on $\mathbf{D}$)

  $\rightsquigarrow$ expected #BM at $C^{(y1)}$ in first partitioning step:
  $\mathbb{E}[(D_1 + D_2) \cdot f(D_1)] \cdot n + O(1)$
- Integrals even more "fun" ... but doable

Branch Misses in YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

- Complication for analysis:
  - 4 branch locations
  - how often they are executed depends on input



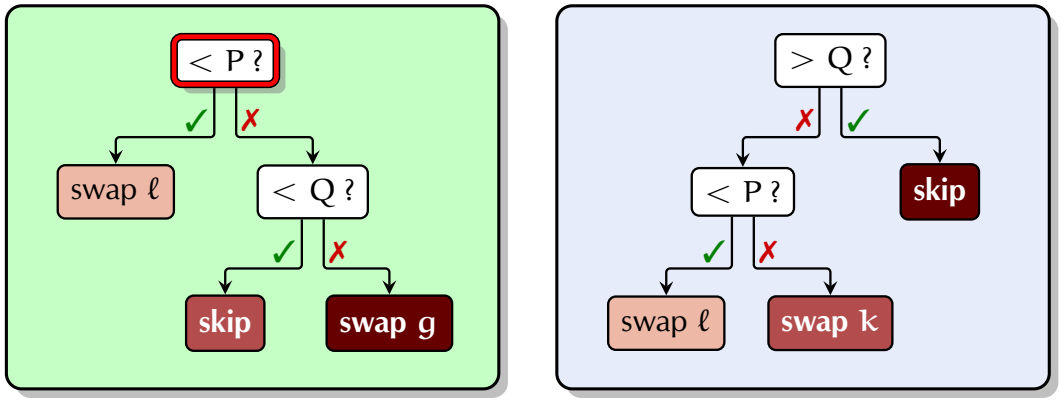
- **Example:** $C^{(y1)}$
  - executed $(\ D_1 + D_2\ )n + O(1)$ times. (in expectation, conditional on $\mathbf{D}$)
  - branch taken i. i. d. with prob D_1. (conditional on $\mathbf{D}$)

  $\rightsquigarrow$ expected #BM at $C^{(y1)}$ in first partitioning step:
  $\mathbb{E}[(D_1 + D_2) \cdot f(D_1)] \cdot n + O(1)$
- Integrals even more "fun" ... but doable

Branch Misses in YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

- Complication for analysis:
  - 4 branch locations
  - how often they are executed depends on input



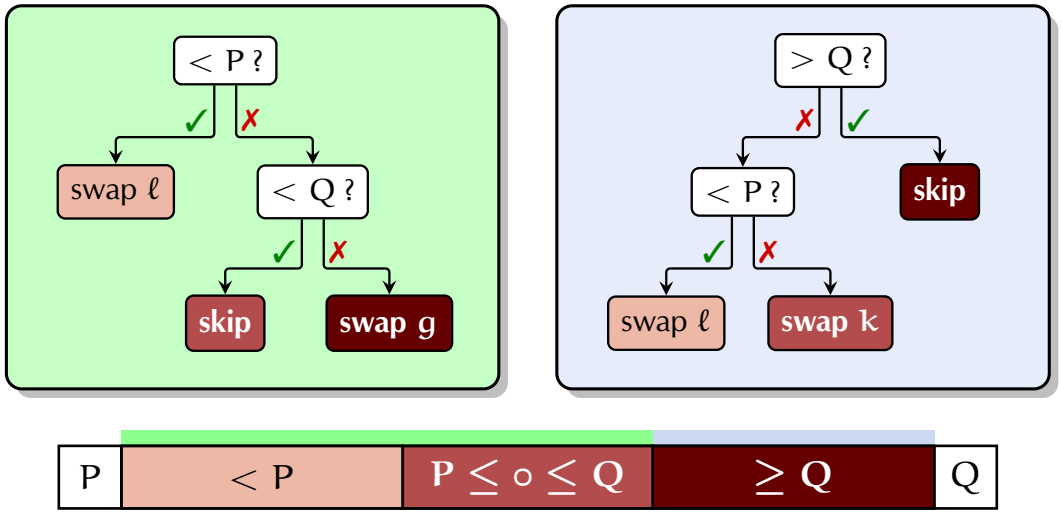
- **Example:** $C^{(y1)}$
  - executed $(D_1 + D_2)n + O(1)$ times. (in expectation, conditional on $\mathbf{D}$)
  - branch taken i. i. d. with prob D_1. (conditional on $\mathbf{D}$)

  $\rightsquigarrow$ expected #BM at $C^{(y1)}$ in first partitioning step:
  $\mathbb{E}[(D_1 + D_2) \cdot f(D_1)] \cdot n + O(1)$
- Integrals even more "fun" ... but doable

Branch Misses in YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

- Complication for analysis:
  - 4 branch locations
  - how often they are executed depends on input



- **Example:** $C^{(y1)}$
  - executed $(D_1 + D_2)n + O(1)$ times. (in expectation, conditional on $\mathbf{D}$)
  - branch taken i. i. d. with prob D_1. (conditional on $\mathbf{D}$)

  $\rightsquigarrow$ expected #BM at $C^{(y1)}$ in first partitioning step:
  $\mathbb{E}[(D_1 + D_2) \cdot f(D_1)] \cdot n + O(1)$
- Integrals even more "fun" ... but doable

Results CQS vs. YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

Expected **number of branch misses**

without pivot sampling

	CQS	YQS	Relative
OPT	0.5	0.513	+2.6%
1-bit	0.6	0.673	+1.0%
2-bit sc	0.571	0.585	+2.5%
2-bit fc	0.589	0.602	+2.2%
	$\cdot n \ln n + O(n)$		

CQS median-of-3 vs. YQS tertiles-of-5

	CQS	YQS	Relative
OPT	0.536	0.538	+0.4%
1-bit	0.686	0.687	+0.1%
2-bit sc	0.611	0.613	+0.3%
2-bit fc	0.627	0.629	+0.3%
	$\cdot n \ln n + O(n)$		

→ essentially same number of BMs

→ branch misses not a plausible explanation for YQS's success

Results CQS vs. YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

Expected **number of branch misses**

without pivot sampling

	CQS	YQS	Relative
OPT	0.5	0.513	+2.6%
1-bit	$0.\overline{6}$	0.673	+1.0%
2-bit sc	0.571	0.585	+2.5%
2-bit fc	0.589	0.602	+2.2%

$\cdot n \ln n + O(n)$

CQS median-of-3 vs. YQS tertiles-of-5

	CQS	YQS	Relative
OPT	0.536	0.538	+0.4%
1-bit	0.686	0.687	+0.1%
2-bit sc	0.611	0.613	+0.3%
2-bit fc	0.627	0.629	+0.3%

$\cdot n \ln n + O(n)$

$\rightsquigarrow$ essentially **same** number of BM.

$\rightsquigarrow$ Branch misses **not** a plausible explanation for YQS's success.

Results CQS vs. YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

Expected **number of branch misses**

without pivot sampling

	CQS	YQS	Relative
OPT	0.5	0.513	+2.6%
1-bit	$0.\overline{6}$	0.673	+1.0%
2-bit sc	0.571	0.585	+2.5%
2-bit fc	0.589	0.602	+2.2%

$\cdot n \ln n + O(n)$

CQS median-of-3 vs. YQS tertiles-of-5

	CQS	YQS	Relative
OPT	0.536	0.538	+0.4%
1-bit	0.686	0.687	+0.1%
2-bit sc	0.611	0.613	+0.3%
2-bit fc	0.627	0.629	+0.3%

$\cdot n \ln n + O(n)$

$\rightsquigarrow$ essentially **same** number of BM.

$\rightsquigarrow$ Branch misses **not** a plausible explanation for YQS's success.

Results CQS vs. YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

Expected **number of branch misses**

without pivot sampling

	CQS	YQS	Relative
OPT	0.5	0.513	+2.6%
1-bit	$0.\overline{6}$	0.673	+1.0%
2-bit sc	0.571	0.585	+2.5%
2-bit fc	0.589	0.602	+2.2%

$\cdot n \ln n + O(n)$

CQS median-of-3 vs. YQS tertiles-of-5

	CQS	YQS	Relative
OPT	0.536	0.538	+0.4%
1-bit	0.686	0.687	+0.1%
2-bit sc	0.611	0.613	+0.3%
2-bit fc	0.627	0.629	+0.3%

$\cdot n \ln n + O(n)$

$\rightsquigarrow$ essentially **same** number of BM.

$\rightsquigarrow$ Branch misses **not** a plausible explanation for YQS's success.

Results CQS vs. YQS

Original question: ***Does YQS better than CQS w. r. t. branch misses?***

Expected **number of branch misses**

without pivot sampling

	CQS	YQS	Relative
OPT	0.5	0.513	+2.6%
1-bit	$0.\overline{6}$	0.673	+1.0%
2-bit sc	0.571	0.585	+2.5%
2-bit fc	0.589	0.602	+2.2%

$\cdot n \ln n + O(n)$

CQS median-of-3 vs. YQS tertiles-of-5

	CQS	YQS	Relative
OPT	0.536	0.538	+0.4%
1-bit	0.686	0.687	+0.1%
2-bit sc	0.611	0.613	+0.3%
2-bit fc	0.627	0.629	+0.3%

$\cdot n \ln n + O(n)$

- ⇝ essentially **same** number of BM.
- ⇝ Branch misses **not** a plausible explanation for YQS's success.

Conclusion

- **Precise analysis** of branch misses in Quicksort (CQS and YQS)
  - including **pivot sampling**
  - lower bounds on branch miss rates

- CQS and YQS cause very **similar** number of BM

⇝ **Strengthened evidence** for the hypothesis that
YQS is faster because of better usage of **memory hierarchy**.

Conclusion

- **Precise analysis** of branch misses in Quicksort (CQS and YQS)
  - including **pivot sampling**
  - lower bounds on branch miss rates
- CQS and YQS cause very **similar** number of BM

$\rightsquigarrow$ **Strengthened evidence** for the hypothesis that
YQS is faster because of better usage of **memory hierarchy**.

Conclusion

- **Precise analysis** of branch misses in Quicksort (CQS and YQS)
  - including **pivot sampling**
  - lower bounds on branch miss rates

- CQS and YQS cause very **similar** number of BM

⇝ **Strengthened evidence** for the hypothesis that
YQS is faster because of better usage of **memory hierarchy**.

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp = 0.25$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp = 0.25$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp = 0.25$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp = 0.\overline{3}$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp = 0.25$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp = 0.\overline{3}$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$

Miss Rate for Branches in Quicksort

- **without sampling:** $P \overset{\mathcal{D}}{=} \mathit{Uniform}(0,1)$
  - $\mathbb{E}[f_{\text{OPT}}(P)] = \int_0^1 \min\{p, 1-p\}\,dp = 0.25$
  - $\mathbb{E}[f_{\text{1-bit}}(P)] = \int_0^1 2p(1-p)\,dp = 0.\overline{3}$
  - $\mathbb{E}[f_{\text{2-bit-sc}}(P)] = \int_0^1 \frac{p(1-p)}{1-2p(1-p)}\,dp = \frac{\pi}{4} - \frac{1}{2} \approx 0.285$
  - $\mathbb{E}[f_{\text{2-bit-fc}}(P)] = \int_0^1 \frac{2p^2(1-p)^2 + p(1-p)}{1-2p(1-p)}\,dp = \frac{2\pi}{\sqrt{3}} - \frac{10}{3} \approx 0.294$