

Algorithms for collaborative text editing

MARTIN KLEPPMANN

until Dec 2023
TU Munich

from Jan 2024
University of Cambridge

email martin@kleppmann.com

web <https://martin.kleppmann.com>

bluesky [@martin.kleppmann.com](https://bluesky.martin.kleppmann.com)

mastodon [@martin@nondeterministic.computer](https://nondeterministic.com/@martin)



VolkswagenStiftung

Ink & Switch

hack on code



time



hack on code

git commit

time





hack on code

git commit

time

GITHUB



git push





hack on code

git commit

time

GITHUB



git push



hack on code

git commit





hack on code

git commit

time

GITHUB



git push



hack on code

git commit

git merge

git fetch

* backend_test_LOCAL_3486.js, backend_test_BASE_3486.js, backend_test_REMOTE_3486.js - SourceGear DiffMerge

```

...martin/dev/cl/automerge/test/backend_test_LOCAL_3486.js ...rtin/dev/cl/automerge/test/backend_test_BASE_3486.js ...martin/dev/cl/automerge/test/backend_test_REMOTE_3486.js

5 const ROOT_ID = '00000000-0000-0000-0000-00000000
6
7 describe('Backend', () => {
8   describe('incremental diffs', () => {
9     it('should assign to a key in a map', () => {
10       const actor = uuid()
11       const changel = {actor, seq: 1, deps: {}},
12         {action: 'set', obj: ROOT_ID, key: 'bird'
13       })
14
15       const s0 = Backend.init()
16       const [s1, patch1] = Backend.applyChanges(
17         assert.deepEqual(patch1, {
18           canUndo: false, canRedo: false, clock: {
19             diffbs: {objectId: ROOT_ID, type: 'map',
20               bird: {[actor]: {value: 'magpie'}}}
21           }
22         })
23
24       it('should increment a key in a map', () =>
25         const actor = uuid()
26         const changel = {actor, seq: 1, deps: {}},
27           {action: 'Automerge.Backend', () => {
28             describe('incremental diffs', () => {
29               it('should assign to a key in a map', () => {
30                 const actor = uuid()
31                 const changel = {actor, seq: 1, deps: {}},
32                   {action: 'set', obj: ROOT_ID,
33                     key: 'bird',
34                     value: 'magpie'}
35                   } as Op,
36                   {action: 'set', obj: ROOT_ID,
37                     path: [],
38                     type: 'map',
39                     key: 'bird',
40                     value: 'magpie'}
41                   } as Diff,
42                 })
43
44       it('should increment a key in a map', () => {
45         const actor = uuid()
46         const changel = {actor, seq: 1,
47           actor,
48           seq: 1,
49         })
50     })
51
52     const s0 = Backend.init()
53     const [s1, patch1] = Backend.applyChanges(s0)
54     assert.deepEqual(patch1, {
55       canUndo: false,
56       canRedo: false,
57       clock: {[actor]: 1},
58       deps: {[actor]: 1},
59       diffbs: [
60         {
61           action: 'set',
62           obj: ROOT_ID,
63           path: [],
64           type: 'map',
65           key: 'bird',
66           value: 'magpie'
67         } as Diff,
68       ],
69     })
70   })
71
72   it('should increment a key in a map', () => {
73     const actor = uuid()
74     const changel = {actor, seq: 1, deps: {}},
75       {action: 'set', obj: ROOT_ID,
76         key: 'bird',
77         value: 'magpie'}
78       } as Op,
79       {action: 'set', obj: ROOT_ID,
80         path: [],
81         type: 'map',
82         key: 'bird',
83         value: 'magpie'}
84       } as Diff,
85     })
86
87     const s0 = Backend.init()
88     const [s1, patch1] = Backend.applyChanges(s0)
89     assert.deepEqual(patch1, {
90       canUndo: false,
91       canRedo: false,
92       clock: {[actor]: 1},
93       deps: {[actor]: 1},
94       diffbs: [
95         {
96           action: 'set',
97           obj: ROOT_ID,
98           path: [],
99           type: 'map',
100          key: 'bird',
101          value: 'magpie'
102        } as Diff,
103      ],
104    })
105  })
106
107  const s0 = Backend.init()
108  const [s1, patch1] = Backend.applyChanges(s0)
109  assert.deepEqual(patch1, {
110    canUndo: false,
111    canRedo: false,
112    clock: {[actor]: 1},
113    deps: {[actor]: 1},
114    diffbs: [
115      {
116        action: 'set',
117        obj: ROOT_ID,
118        path: [],
119        type: 'map',
120        key: 'bird',
121        value: 'magpie'
122      } as Diff,
123    ],
124  })
125
126  it('should increment a key in a map', () => {
127    const actor = uuid()
128    const changel = {actor, seq: 1, deps: {}},
129      {action: 'set', obj: ROOT_ID,
130        key: 'bird',
131        value: 'magpie'}
132        } as Op,
133        {action: 'set', obj: ROOT_ID,
134          path: [],
135          type: 'map',
136          key: 'bird',
137          value: 'magpie'}
138        } as Diff,
139      })
140
141      const s0 = Backend.init()
142      const [s1, patch1] = Backend.applyChanges(s0)
143      assert.deepEqual(patch1, {
144        canUndo: false,
145        canRedo: false,
146        clock: {[actor]: 1},
147        deps: {[actor]: 1},
148        diffbs: [
149          {
150            action: 'set',
151            obj: ROOT_ID,
152            path: [],
153            type: 'map',
154            key: 'bird',
155            value: 'magpie'
156          } as Diff,
157        ],
158      })
159    })
160
161    const s0 = Backend.init()
162    const [s1, patch1] = Backend.applyChanges(s0)
163    assert.deepEqual(patch1, {
164      canUndo: false,
165      canRedo: false,
166      clock: {[actor]: 1},
167      deps: {[actor]: 1},
168      diffbs: [
169        {
170          action: 'set',
171          obj: ROOT_ID,
172          path: [],
173          type: 'map',
174          key: 'bird',
175          value: 'magpie'
176        } as Diff,
177      ],
178    })
179
180    it('should increment a key in a map', () => {
181      const actor = uuid()
182      const changel = {actor, seq: 1, deps: {}},
183        {action: 'set', obj: ROOT_ID,
184          key: 'bird',
185          value: 'magpie'}
186        } as Op,
187        {action: 'set', obj: ROOT_ID,
188          path: [],
189          type: 'map',
190          key: 'bird',
191          value: 'magpie'}
192        } as Diff,
193      })
194
195      const s0 = Backend.init()
196      const [s1, patch1] = Backend.applyChanges(s0)
197      assert.deepEqual(patch1, {
198        canUndo: false,
199        canRedo: false,
200        clock: {[actor]: 1},
201        deps: {[actor]: 1},
202        diffbs: [
203          {
204            action: 'set',
205            obj: ROOT_ID,
206            path: [],
207            type: 'map',
208            key: 'bird',
209            value: 'magpie'
210          } as Diff,
211        ],
212      })
213
214      it('should increment a key in a map', () => {
215        const actor = uuid()
216        const changel = {actor, seq: 1, deps: {}},
217          {action: 'set', obj: ROOT_ID,
218            key: 'bird',
219            value: 'magpie'}
220            } as Op,
221            {action: 'set', obj: ROOT_ID,
222              path: [],
223              type: 'map',
224              key: 'bird',
225              value: 'magpie'}
226            } as Diff,
227          })
228
229          const s0 = Backend.init()
230          const [s1, patch1] = Backend.applyChanges(s0)
231          assert.deepEqual(patch1, {
232            canUndo: false,
233            canRedo: false,
234            clock: {[actor]: 1},
235            deps: {[actor]: 1},
236            diffbs: [
237              {
238                action: 'set',
239                obj: ROOT_ID,
240                path: [],
241                type: 'map',
242                key: 'bird',
243                value: 'magpie'
244              } as Diff,
245            ],
246          })
247
248          it('should increment a key in a map', () => {
249            const actor = uuid()
250            const changel = {actor, seq: 1, deps: {}},
251              {action: 'set', obj: ROOT_ID,
252                key: 'bird',
253                value: 'magpie'}
254                } as Op,
255                {action: 'set', obj: ROOT_ID,
256                  path: [],
257                  type: 'map',
258                  key: 'bird',
259                  value: 'magpie'}
260                } as Diff,
261              })
262
263              const s0 = Backend.init()
264              const [s1, patch1] = Backend.applyChanges(s0)
265              assert.deepEqual(patch1, {
266                canUndo: false,
267                canRedo: false,
268                clock: {[actor]: 1},
269                deps: {[actor]: 1},
270                diffbs: [
271                  {
272                    action: 'set',
273                    obj: ROOT_ID,
274                    path: [],
275                    type: 'map',
276                    key: 'bird',
277                    value: 'magpie'
278                  } as Diff,
279                ],
280              })
281
282              it('should increment a key in a map', () => {
283                const actor = uuid()
284                const changel = {actor, seq: 1, deps: {}},
285                  {action: 'set', obj: ROOT_ID,
286                    key: 'bird',
287                    value: 'magpie'}
288                    } as Op,
289                    {action: 'set', obj: ROOT_ID,
290                      path: [],
291                      type: 'map',
292                      key: 'bird',
293                      value: 'magpie'}
294                    } as Diff,
295                  })
296
297                  const s0 = Backend.init()
298                  const [s1, patch1] = Backend.applyChanges(s0)
299                  assert.deepEqual(patch1, {
300                    canUndo: false,
301                    canRedo: false,
302                    clock: {[actor]: 1},
303                    deps: {[actor]: 1},
304                    diffbs: [
305                      {
306                        action: 'set',
307                        obj: ROOT_ID,
308                        path: [],
309                        type: 'map',
310                        key: 'bird',
311                        value: 'magpie'
312                      } as Diff,
313                    ],
314                  })
315
316                  it('should increment a key in a map', () => {
317                    const actor = uuid()
318                    const changel = {actor, seq: 1, deps: {}},
319                      {action: 'set', obj: ROOT_ID,
320                        key: 'bird',
321                        value: 'magpie'}
322                        } as Op,
323                        {action: 'set', obj: ROOT_ID,
324                          path: [],
325                          type: 'map',
326                          key: 'bird',
327                          value: 'magpie'}
328                        } as Diff,
329                      })
330
331                      const s0 = Backend.init()
332                      const [s1, patch1] = Backend.applyChanges(s0)
333                      assert.deepEqual(patch1, {
334                        canUndo: false,
335                        canRedo: false,
336                        clock: {[actor]: 1},
337                        deps: {[actor]: 1},
338                        diffbs: [
339                          {
340                            action: 'set',
341                            obj: ROOT_ID,
342                            path: [],
343                            type: 'map',
344                            key: 'bird',
345                            value: 'magpie'
346                          } as Diff,
347                        ],
348                      })
349
350                      it('should increment a key in a map', () => {
351                        const actor = uuid()
352                        const changel = {actor, seq: 1, deps: {}},
353                          {action: 'set', obj: ROOT_ID,
354                            key: 'bird',
355                            value: 'magpie'}
356                            } as Op,
357                            {action: 'set', obj: ROOT_ID,
358                              path: [],
359                              type: 'map',
360                              key: 'bird',
361                              value: 'magpie'}
362                            } as Diff,
363                          })
364
365                          const s0 = Backend.init()
366                          const [s1, patch1] = Backend.applyChanges(s0)
367                          assert.deepEqual(patch1, {
368                            canUndo: false,
369                            canRedo: false,
370                            clock: {[actor]: 1},
371                            deps: {[actor]: 1},
372                            diffbs: [
373                              {
374                                action: 'set',
375                                obj: ROOT_ID,
376                                path: [],
377                                type: 'map',
378                                key: 'bird',
379                                value: 'magpie'
380                              } as Diff,
381                            ],
382                          })
383
384                          it('should increment a key in a map', () => {
385                            const actor = uuid()
386                            const changel = {actor, seq: 1, deps: {}},
387                              {action: 'set', obj: ROOT_ID,
388                                key: 'bird',
389                                value: 'magpie'}
390                                } as Op,
391                                {action: 'set', obj: ROOT_ID,
392                                  path: [],
393                                  type: 'map',
394                                  key: 'bird',
395                                  value: 'magpie'}
396                                } as Diff,
397                              })
398
399                              const s0 = Backend.init()
400                              const [s1, patch1] = Backend.applyChanges(s0)
401                              assert.deepEqual(patch1, {
402                                canUndo: false,
403                                canRedo: false,
404                                clock: {[actor]: 1},
405                                deps: {[actor]: 1},
406                                diffbs: [
407                                  {
408                                    action: 'set',
409                                    obj: ROOT_ID,
410                                    path: [],
411                                    type: 'map',
412                                    key: 'bird',
413                                    value: 'magpie'
414                                  } as Diff,
415                                ],
416                              })
417
418                              it('should increment a key in a map', () => {
419                                const actor = uuid()
420                                const changel = {actor, seq: 1, deps: {}},
421                                  {action: 'set', obj: ROOT_ID,
422                                    key: 'bird',
423                                    value: 'magpie'}
424                                    } as Op,
425                                    {action: 'set', obj: ROOT_ID,
426                                      path: [],
427                                      type: 'map',
428                                      key: 'bird',
429                                      value: 'magpie'}
430                                    } as Diff,
431                                  })
432
433                                  const s0 = Backend.init()
434                                  const [s1, patch1] = Backend.applyChanges(s0)
435                                  assert.deepEqual(patch1, {
436                                    canUndo: false,
437                                    canRedo: false,
438                                    clock: {[actor]: 1},
439                                    deps: {[actor]: 1},
440                                    diffbs: [
441                                      {
442                                        action: 'set',
443                                        obj: ROOT_ID,
444                                        path: [],
445                                        type: 'map',
446                                        key: 'bird',
447                                        value: 'magpie'
448                                      } as Diff,
449                                    ],
450                                  })
451
452                                  it('should increment a key in a map', () => {
453                                    const actor = uuid()
454                                    const changel = {actor, seq: 1, deps: {}},
455                                      {action: 'set', obj: ROOT_ID,
456                                        key: 'bird',
457                                        value: 'magpie'}
458                                        } as Op,
459                                        {action: 'set', obj: ROOT_ID,
460                                          path: [],
461                                          type: 'map',
462                                          key: 'bird',
463                                          value: 'magpie'}
464                                        } as Diff,
465                                      })
466
467                                      const s0 = Backend.init()
468                                      const [s1, patch1] = Backend.applyChanges(s0)
469                                      assert.deepEqual(patch1, {
470                                        canUndo: false,
471                                        canRedo: false,
472                                        clock: {[actor]: 1},
473                                        deps: {[actor]: 1},
474                                        diffbs: [
475                                          {
476                                            action: 'set',
477                                            obj: ROOT_ID,
478                                            path: [],
479                                            type: 'map',
480                                            key: 'bird',
481                                            value: 'magpie'
482                                          } as Diff,
483                                        ],
484                                      })
485
486                                      it('should increment a key in a map', () => {
487                                        const actor = uuid()
488                                        const changel = {actor, seq: 1, deps: {}},
489                                          {action: 'set', obj: ROOT_ID,
490                                            key: 'bird',
491                                            value: 'magpie'}
492                                            } as Op,
493                                            {action: 'set', obj: ROOT_ID,
494                                              path: [],
495                                              type: 'map',
496                                              key: 'bird',
497                                              value: 'magpie'}
498                                            } as Diff,
499                                          })
500
501                                          const s0 = Backend.init()
502                                          const [s1, patch1] = Backend.applyChanges(s0)
503                                          assert.deepEqual(patch1, {
504                                            canUndo: false,
505                                            canRedo: false,
506                                            clock: {[actor]: 1},
507                                            deps: {[actor]: 1},
508                                            diffbs: [
509                                              {
510                                                action: 'set',
511                                                obj: ROOT_ID,
512                                                path: [],
513                                                type: 'map',
514                                                key: 'bird',
515                                                value: 'magpie'
516                                              } as Diff,
517                                            ],
518                                          })
519
520                                          it('should increment a key in a map', () => {
521                                            const actor = uuid()
522                                            const changel = {actor, seq: 1, deps: {}},
523                                              {action: 'set', obj: ROOT_ID,
524                                                key: 'bird',
525                                                value: 'magpie'}
526                                                } as Op,
527                                                {action: 'set', obj: ROOT_ID,
528                                                  path: [],
529                                                  type: 'map',
530                                                  key: 'bird',
531                                                  value: 'magpie'}
532                                                } as Diff,
533                                              })
534
535                                              const s0 = Backend.init()
536                                              const [s1, patch1] = Backend.applyChanges(s0)
537                                              assert.deepEqual(patch1, {
538                                                canUndo: false,
539                                                canRedo: false,
540                                                clock: {[actor]: 1},
541                                                deps: {[actor]: 1},
542                                                diffbs: [
543                                                  {
544                                                    action: 'set',
545                                                    obj: ROOT_ID,
546                                                    path: [],
547                                                    type: 'map',
548                                                    key: 'bird',
549                                                    value: 'magpie'
550                                                  } as Diff,
551                                                ],
552                                              })
553
554                                              it('should increment a key in a map', () => {
555                                                const actor = uuid()
556                                                const changel = {actor, seq: 1, deps: {}},
557                                                  {action: 'set', obj: ROOT_ID,
558                                                    key: 'bird',
559                                                    value: 'magpie'}
560                                                    } as Op,
561                                                    {action: 'set', obj: ROOT_ID,
562                                                      path: [],
563                                                      type: 'map',
564                                                      key: 'bird',
565                                                      value: 'magpie'}
566                                                    } as Diff,
567                                                  })
568
569                                                  const s0 = Backend.init()
570                                                  const [s1, patch1] = Backend.applyChanges(s0)
571                                                  assert.deepEqual(patch1, {
572                                                    canUndo: false,
573                                                    canRedo: false,
574                                                    clock: {[actor]: 1},
575                                                    deps: {[actor]: 1},
576                                                    diffbs: [
577                                                      {
578                                                        action: 'set',
579                                                        obj: ROOT_ID,
580                                                        path: [],
581                                                        type: 'map',
582                                                        key: 'bird',
583                                                        value: 'magpie'
584                                                      } as Diff,
585                                                    ],
586                                                  })
587
588                                                  it('should increment a key in a map', () => {
589                                                    const actor = uuid()
590                                                    const changel = {actor, seq: 1, deps: {}},
591                                                      {action: 'set', obj: ROOT_ID,
592                                                        key: 'bird',
593                                                        value: 'magpie'}
594                                                        } as Op,
595                                                        {action: 'set', obj: ROOT_ID,
596                                                          path: [],
597                                                          type: 'map',
598                                                          key: 'bird',
599                                                          value: 'magpie'}
600                                                        } as Diff,
601                                                      })
602
603                                                      const s0 = Backend.init()
604                                                      const [s1, patch1] = Backend.applyChanges(s0)
605                                                      assert.deepEqual(patch1, {
606                                                        canUndo: false,
607                                                        canRedo: false,
608                                                        clock: {[actor]: 1},
609                                                        deps: {[actor]: 1},
610                                                        diffbs: [
611                                                          {
612                                                            action: 'set',
613                                                            obj: ROOT_ID,
614                                                            path: [],
615                                                            type: 'map',
616                                                            key: 'bird',
617                                                            value: 'magpie'
618                                                          } as Diff,
619                                                        ],
620                                                      })
621
622                                                      it('should increment a key in a map', () => {
623                                                        const actor = uuid()
624                                                        const changel = {actor, seq: 1, deps: {}},
625                                                          {action: 'set', obj: ROOT_ID,
626                                                            key: 'bird',
627                                                            value: 'magpie'}
628                                                            } as Op,
629                                                            {action: 'set', obj: ROOT_ID,
630                                                              path: [],
631                                                              type: 'map',
632                                                              key: 'bird',
633                                                              value: 'magpie'}
634                                                            } as Diff,
635                                                          })
636
637                                                          const s0 = Backend.init()
638                                                          const [s1, patch1] = Backend.applyChanges(s0)
639                                                          assert.deepEqual(patch1, {
640                                                            canUndo: false,
641                                                            canRedo: false,
642                                                            clock: {[actor]: 1},
643                                                            deps: {[actor]: 1},
644                                                            diffbs: [
645                                                              {
646                                                                action: 'set',
647                                                                obj: ROOT_ID,
648                                                                path: [],
649                                                                type: 'map',
650                                                                key: 'bird',
651                                                                value: 'magpie'
652                                                              } as Diff,
653                                                            ],
654                                                          })
655
656                                                          it('should increment a key in a map', () => {
657                                                            const actor = uuid()
658                                                            const changel = {actor, seq: 1, deps: {}},
659                                                              {action: 'set', obj: ROOT_ID,
660                                                                key: 'bird',
661                                                                value: 'magpie'}
662                                                                } as Op,
663                                                                {action: 'set', obj: ROOT_ID,
664                                                                  path: [],
665                                                                  type: 'map',
666                                                                  key: 'bird',
667                                                                  value: 'magpie'}
668                                                                } as Diff,
669                                                              })
670
671                                                              const s0 = Backend.init()
672                                                              const [s1, patch1] = Backend.applyChanges(s0)
673                                                              assert.deepEqual(patch1, {
674                                                                canUndo: false,
675                                                                canRedo: false,
676                                                                clock: {[actor]: 1},
677                                                                deps: {[actor]: 1},
678                                                                diffbs: [
679                                                                  {
680                                                                    action: 'set',
681                                                                    obj: ROOT_ID,
682                                                                    path: [],
683                                                                    type: 'map',
684                                                                    key: 'bird',
685                                                                    value: 'magpie'
686                                                                  } as Diff,
687                                                                ],
688                                                              })
689
690                                                              it('should increment a key in a map', () => {
691                                                                const actor = uuid()
692                                                                const changel = {actor, seq: 1, deps: {}},
693                                                                  {action: 'set', obj: ROOT_ID,
694                                                                    key: 'bird',
695                                                                    value: 'magpie'}
696                                                                    } as Op,
697                                                                    {action: 'set', obj: ROOT_ID,
698                                                                      path: [],
699                                                                      type: 'map',
700                                                                      key: 'bird',
701                                                                      value: 'magpie'}
702                                                                    } as Diff,
703                                                                  })
704
705                                                                  const s0 = Backend.init()
706                                                                  const [s1, patch1] = Backend.applyChanges(s0)
707                                                                  assert.deepEqual(patch1, {
708                                                                    canUndo: false,
709                                                                    canRedo: false,
710                                                                    clock: {[actor]: 1},
711                                                                    deps: {[actor]: 1},
712                                                                    diffbs: [
713                                                                      {
714                                                                        action: 'set',
715                                                                        obj: ROOT_ID,
716                                                                        path: [],
717                                                                        type: 'map',
718                                                                        key: 'bird',
719                                                                        value: 'magpie'
720                                                                      } as Diff,
721                                                                    ],
722                                                                  })
723
724                                                                  it('should increment a key in a map', () => {
725                                                                    const actor = uuid()
726                                                                    const changel = {actor, seq: 1, deps: {}},
727                                                                      {action: 'set', obj: ROOT_ID,
728                                                                        key: 'bird',
729                                                                        value: 'magpie'}
730                                                                        } as Op,
731                                                                        {action: 'set', obj: ROOT_ID,
732                                                                          path: [],
733                                                                          type: 'map',
734                                                                          key: 'bird',
735                                                                          value: 'magpie'}
736                                                                        } as Diff,
737                                                                      })
738
739                                                                      const s0 = Backend.init()
740                                                                      const [s1, patch1] = Backend.applyChanges(s0)
741                                                                      assert.deepEqual(patch1, {
742                                                                        canUndo: false,
743                                                                        canRedo: false,
744                                                                        clock: {[actor]: 1},
745                                                                        deps: {[actor]: 1},
746                                                                        diffbs: [
747                                                                          {
748                                                                            action: 'set',
749                                                                            obj: ROOT_ID,
750                                                                            path: [],
751                                                                            type: 'map',
752                                                                            key: 'bird',
753                                                                            value: 'magpie'
754                                                                          } as Diff,
755                                                                        ],
756                                                                      })
757
758                                                                      it('should increment a key in a map', () => {
759                                                                        const actor = uuid()
760                                                                        const changel = {actor, seq: 1, deps: {}},
761                                                                          {action: 'set', obj: ROOT_ID,
762                                                                            key: 'bird',
763                                                                            value: 'magpie'}
764                                                                            } as Op,
765                                                                            {action: 'set', obj: ROOT_ID,
766                                                                              path: [],
767                                                                              type: 'map',
768                                                                              key: 'bird',
769                                                                              value: 'magpie'}
770                                                                            } as Diff,
771                                                                          })
772
773                                                                          const s0 = Backend.init()
774                                                                          const [s1, patch1] = Backend.applyChanges(s0)
775                                                                          assert.deepEqual(patch1, {
776                                                                            canUndo: false,
777                                                                            canRedo: false,
778                                                                            clock: {[actor]: 1},
779                                                                            deps: {[actor]: 1},
780                                                                            diffbs: [
781                                                                              {
782                                                                                action: 'set',
783                                                                                obj: ROOT_ID,
784                                                                                path: [],
785                                                                                type: 'map',
786                                                                                key: 'bird',
787                                                                                value: 'magpie'
788                                                                              } as Diff,
789                                                                            ],
790                                                                          })
791
792                                                                          it('should increment a key in a map', () => {
793                                                                            const actor = uuid()
794                                                                            const changel = {actor, seq: 1, deps: {}},
795                                                                              {action: 'set', obj: ROOT_ID,
796                                                                                key: 'bird',
797                                                                                value: 'magpie'}
798                                                                                } as Op,
799                                                                                {action: 'set', obj: ROOT_ID,
800                                                                                  path: [],
801                                                                                  type: 'map',
802                                                                                  key: 'bird',
803                                                                                  value: 'magpie'}
804                                                                                } as Diff,
805                                                                                  })
806
807                                                                                  const s0 = Backend.init()
808                                                                                  const [s1, patch1] = Backend.applyChanges(s0)
809                                                                                  assert.deepEqual(patch1, {
810                                                                                    canUndo: false,
811                                                                                    canRedo: false,
812                                                                                    clock: {[actor]: 1},
813                                                                                    deps: {[actor]: 1},
814                                                                                    diffbs: [
815                                                                                      {
816                                                                                        action: 'set',
817                                                                                        obj: ROOT_ID,
818                                                                                        path: [],
819                                                                                        type: 'map',
820                                                                                        key: 'bird',
821                                                                                        value: 'magpie'
822                                                                                      } as Diff,
823                                                                                    ],
824                                                                                  })
825
826                                                                                  it('should increment a key in a map', () => {
827                                                                                    const actor = uuid()
828                                                                                    const changel = {actor, seq: 1, deps: {}},
829                                                                                      {action: 'set', obj: ROOT_ID,
830                                                                                        key: 'bird',
831                                                                                        value: 'magpie'}
832                                                                                        } as Op,
833                                                                                        {action: 'set', obj: ROOT_ID,
834              
```

Changes: 17; Conflicts: 35

Reference View (Files as Loaded) Edit View (Merge Result)

Ruleset: _Default_ default

Edit Word
document

Save

time



Edit Word
document

Save

time



Edit Word
document

Save

time

Edit Word
document



Save

time

Email (SMTP)

MAIL
SERVER



Edit Word
document

Save

time

Edit Word document



Save

time

MAIL SERVER



Email (SMTP)



Edit Word document

Save

oh crap...

Email (IMAP)

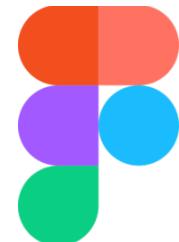
COLLABORATIVE APPLICATIONS



Google Docs



Office 365



Figma

Wide range of domain-specific collaboration software, e.g. for investigative journalism, medical records, data analysis, engineering/CAD, ...

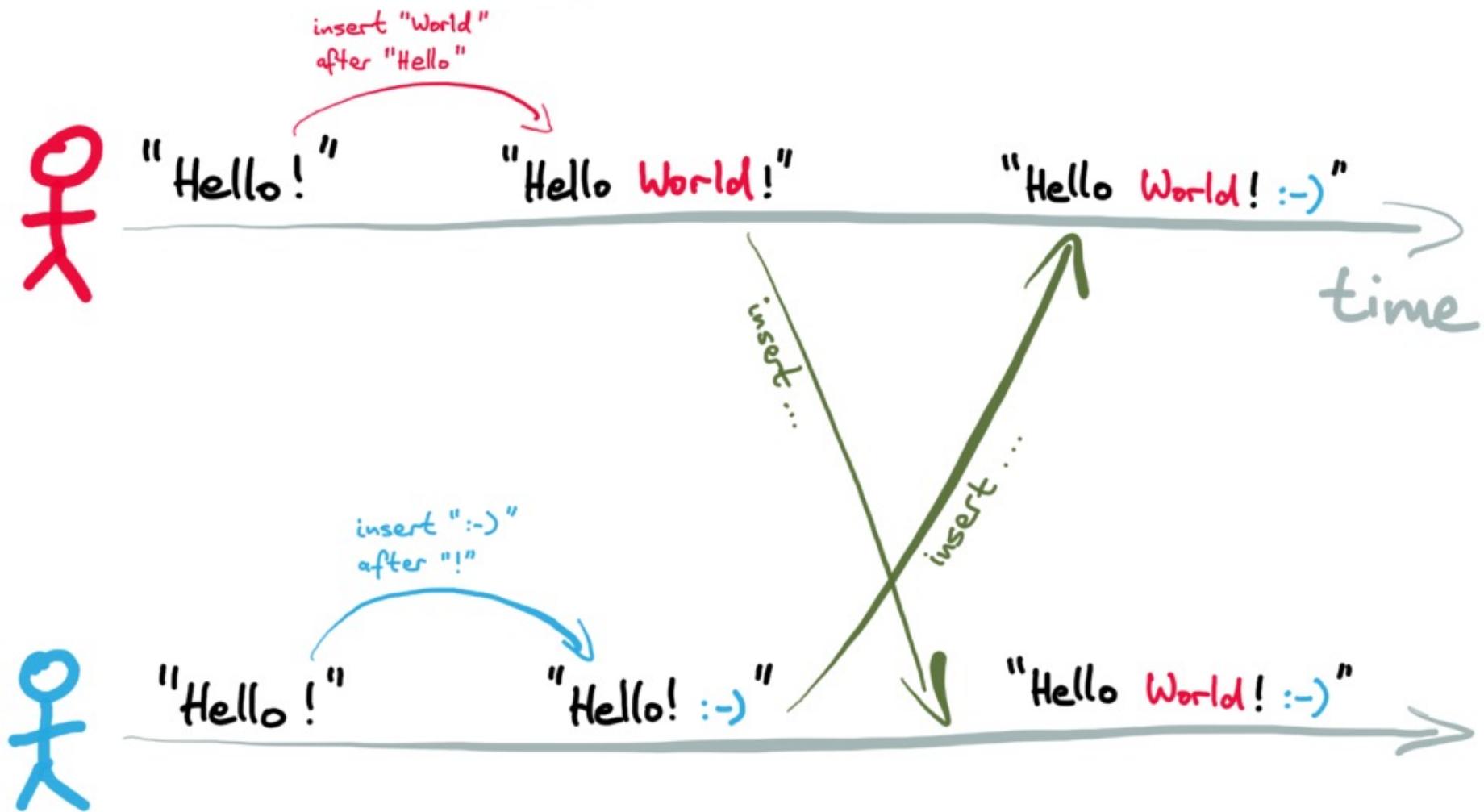
Example: Text editing



Example : Text editing



Example : Text editing



LIVE

DEMO!

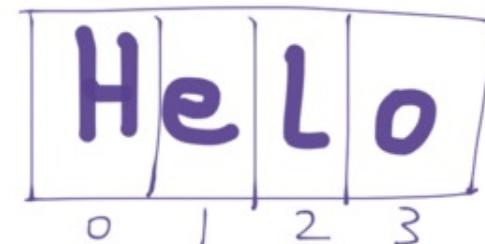
COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CROTs
3. Automerge
4. Rich text – Peritext
5. Non-interleaving – Fugue

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text – Peritext
5. Non-interleaving – Fugue

GOOGLE DOCS (NUTSHELL)



GOOGLE DOCS (NUTSHELL)

H	e	l	o
0	1	2	3

 edit

H	e	l	l	o
0	1	2	3	4

H	e	L	o
0	1	2	3

 edit

H	e	l	o	!
0	1	2	3	4

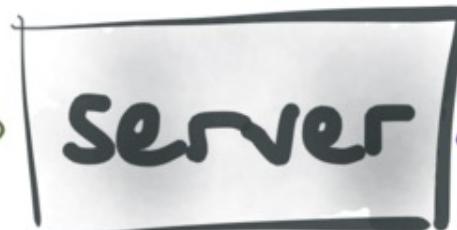
GOOGLE DOCS (NUTSHELL)

H	e	l	o	
0	1	2	3	

edit

H	e	l	l	o
0	1	2	3	4

insert "L"
at pos 3



H	e	l	o	
0	1	2	3	

edit

H	e	l	o	!
0	1	2	3	4

insert "!" at
position 4

GOOGLE DOCS (NUTSHELL)

H	e	l	o	
0	1	2	3	

edit

H	e	l	l	o
0	1	2	3	4

insert "L"
at pos 3



H	e	l	o	
0	1	2	3	

edit

H	e	l	o	!
0	1	2	3	4

insert "!" at
position 4

insert "L" at
position 3

H	e	l	l	o	!
0	1	2	3	4	5

GOOGLE DOCS (NUTSHELL)

H	e	l	o	
0	1	2	3	

edit

H	e	l	l	o	
0	1	2	3	4	

insert "L"
at pos 3



✓ insert "!"
at position 5 (!)

H	e	l	L	l	o	!	
0	1	2	3	4	5		

H	e	l	o	
0	1	2	3	

edit

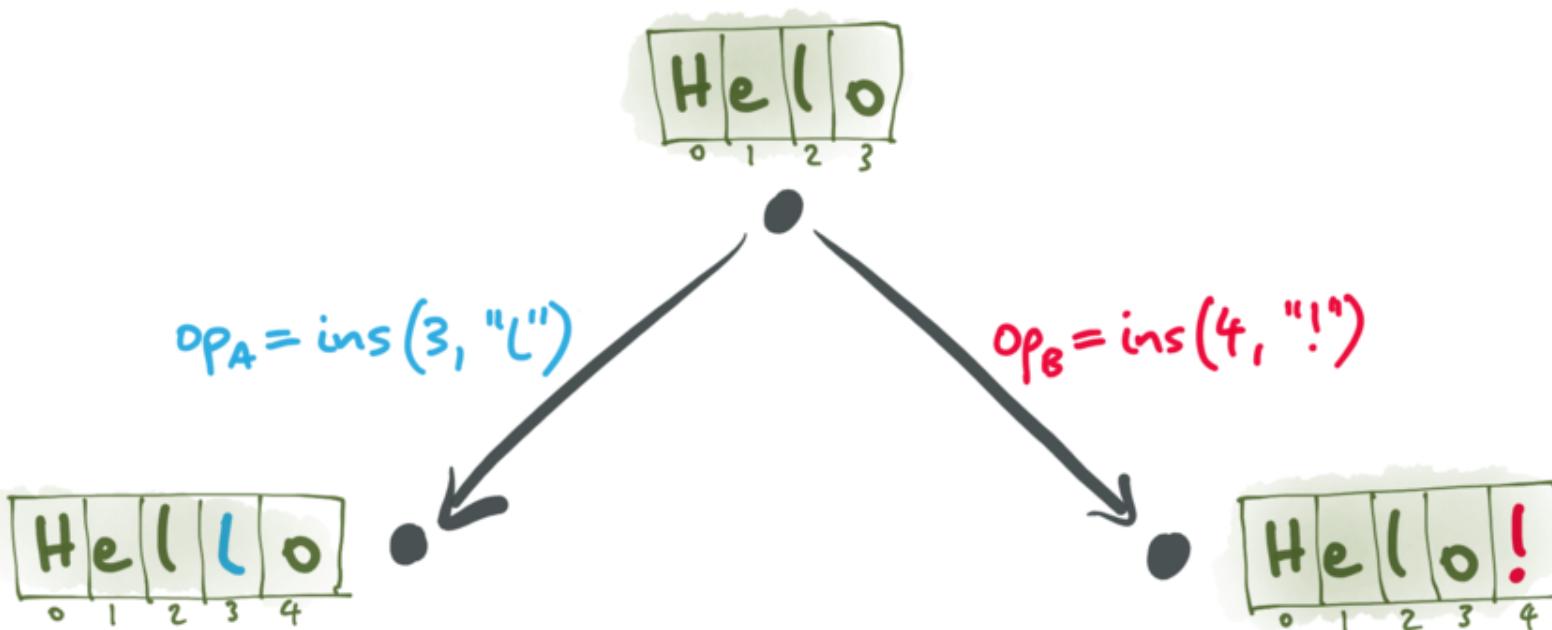
H	e	l	o	!	
0	1	2	3	4	

insert "!" at
position 4

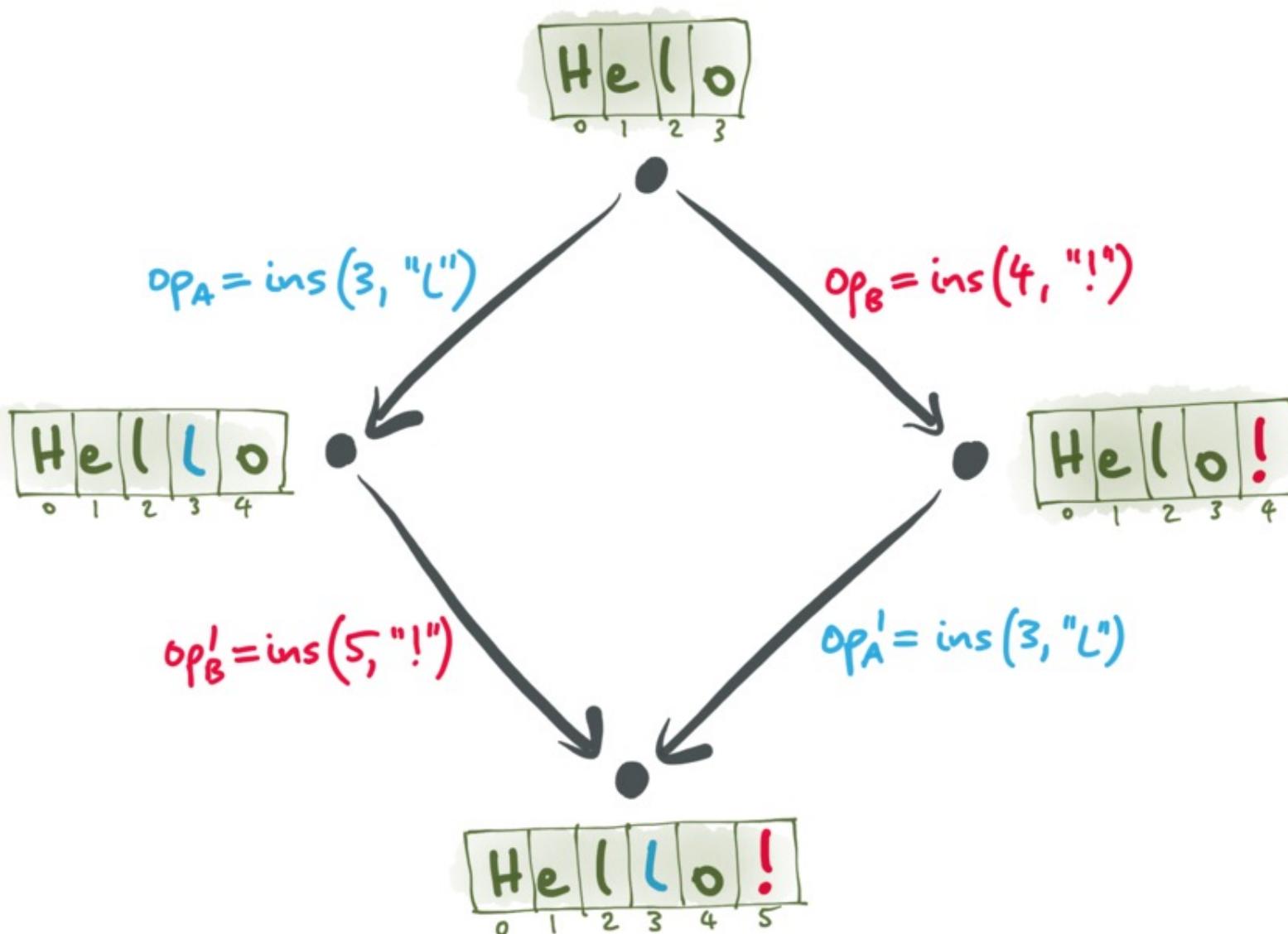
✓ insert "L" at
position 3

H	e	l	L	l	o	!	
0	1	2	3	4	5		

Operational Transformation (OT)



Operational Transformation (OT)



Jupiter: client-server operational transformation (1995)

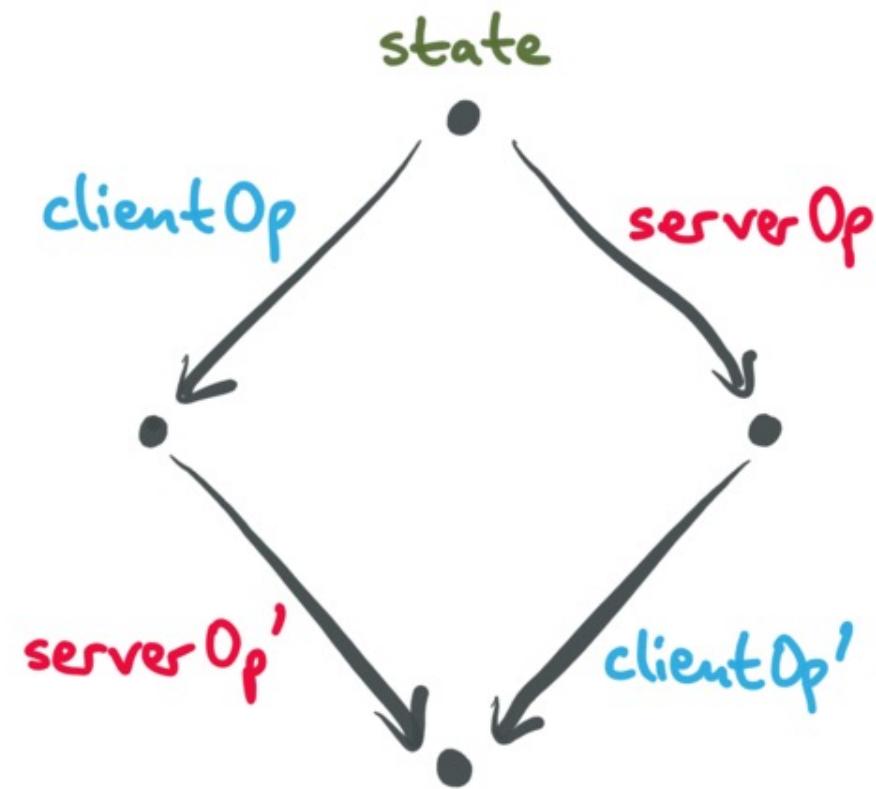
$$xform(\text{clientOp}, \text{serverOp}) = (\text{clientOp}', \text{serverOp}')$$

TP1 property:

Let clientOp and serverOp be concurrent operations that both took place in the same state. Let $xform(\text{clientOp}, \text{serverOp}) = (\text{clientOp}', \text{serverOp}')$.

Then

$$\text{serverOp}'(\text{clientOp}(\text{state})) = \text{clientOp}'(\text{serverOp}(\text{state})).$$



$$xform(\text{ins}(i, x), \text{ins}(j, y)) = \begin{cases} (\text{ins}(i, x), \text{ins}(j+1, y)) & \text{if } i < j \\ (\text{ins}(i+1, x), \text{ins}(j, y)) & \text{if } i \geq j \end{cases}$$

$$xform(\text{ins}(i, x), \text{del}(j)) = \begin{cases} (\text{ins}(i, x), \text{del}(j+1)) & \text{if } i \leq j \\ (\text{ins}(i-1, x), \text{del}(j)) & \text{if } i > j \end{cases}$$

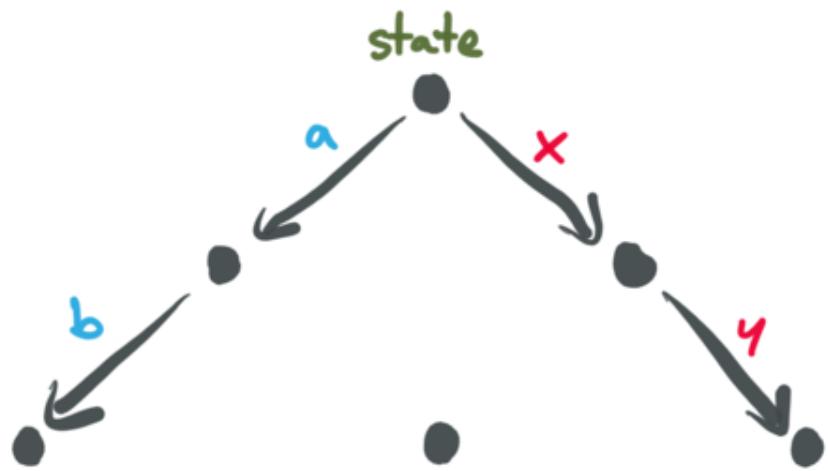
$$xform(\text{del}(i), \text{ins}(j, y)) = \begin{cases} (\text{del}(i), \text{ins}(j-1, y)) & \text{if } i < j \\ (\text{del}(i+1), \text{ins}(j, y)) & \text{if } i \geq j \end{cases}$$

$$xform(\text{del}(i), \text{del}(j)) = \begin{cases} (\text{del}(i), \text{del}(j-1)) & \text{if } i < j \\ (\text{del}(i-1), \text{del}(j)) & \text{if } i > j \\ (\text{noop}, \text{noop}) & \text{if } i = j \end{cases}$$

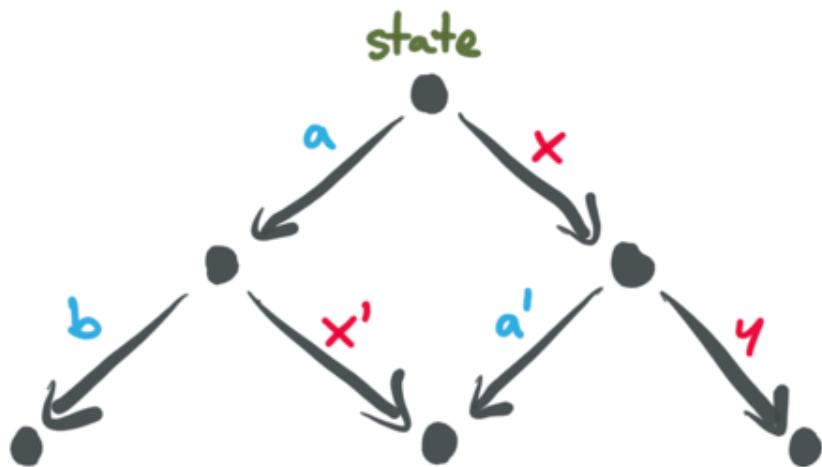
$$xform(\text{op}, \text{noop}) = (\text{op}, \text{noop})$$

$$xform(\text{noop}, \text{op}) = (\text{noop}, \text{op})$$

Diverged by more than one op?

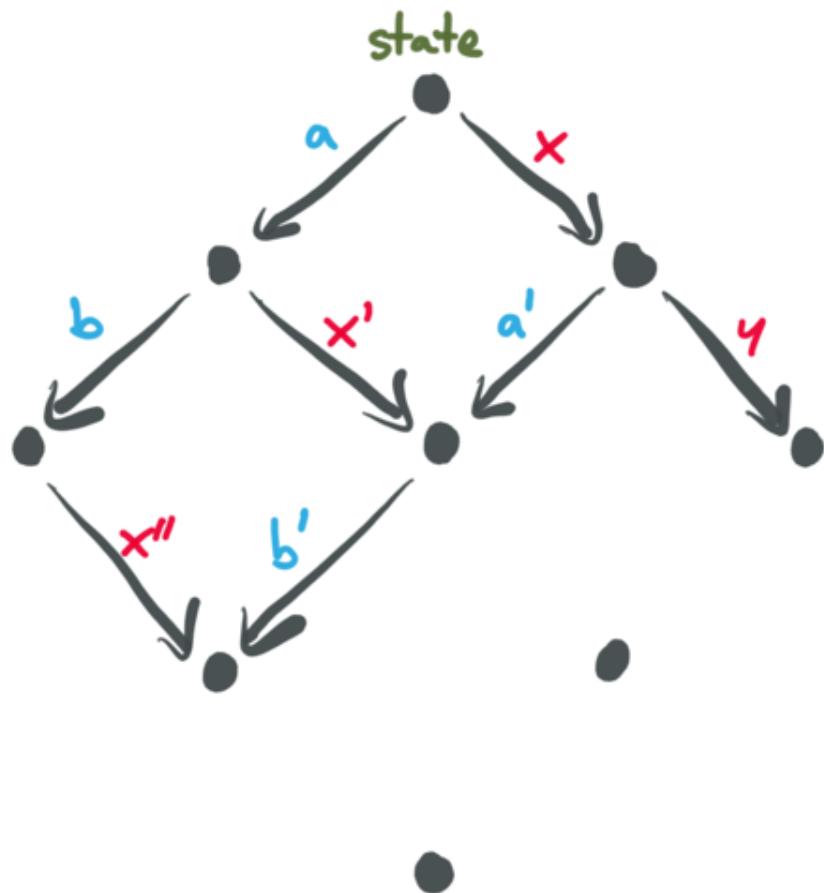


Diverged by more than one op?



$$\text{xform}(a, x) = (a', x')$$

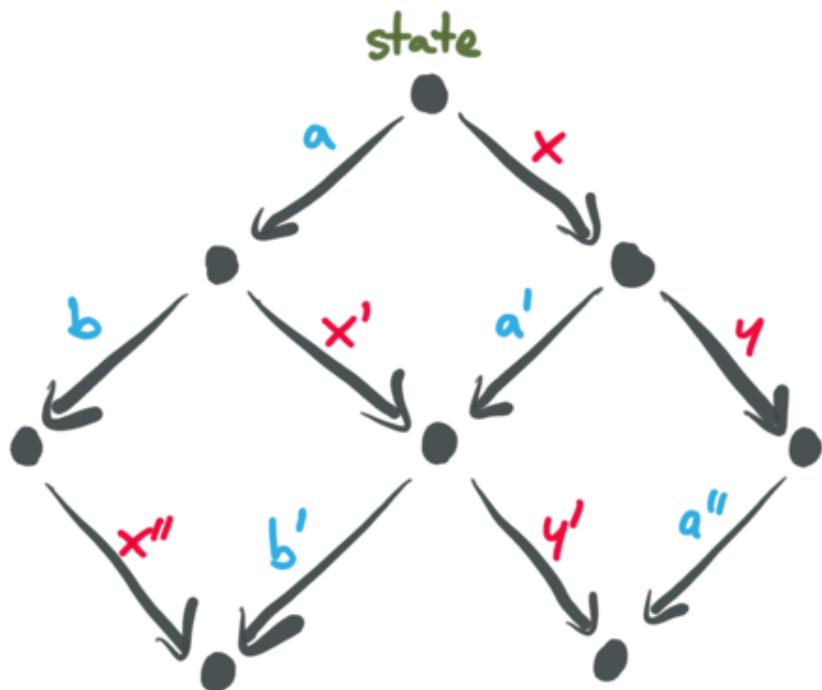
Diverged by more than one op?



$$xform(a, x) = (a', x')$$

$$xform(b, x') = (b', x'')$$

Diverged by more than one op?

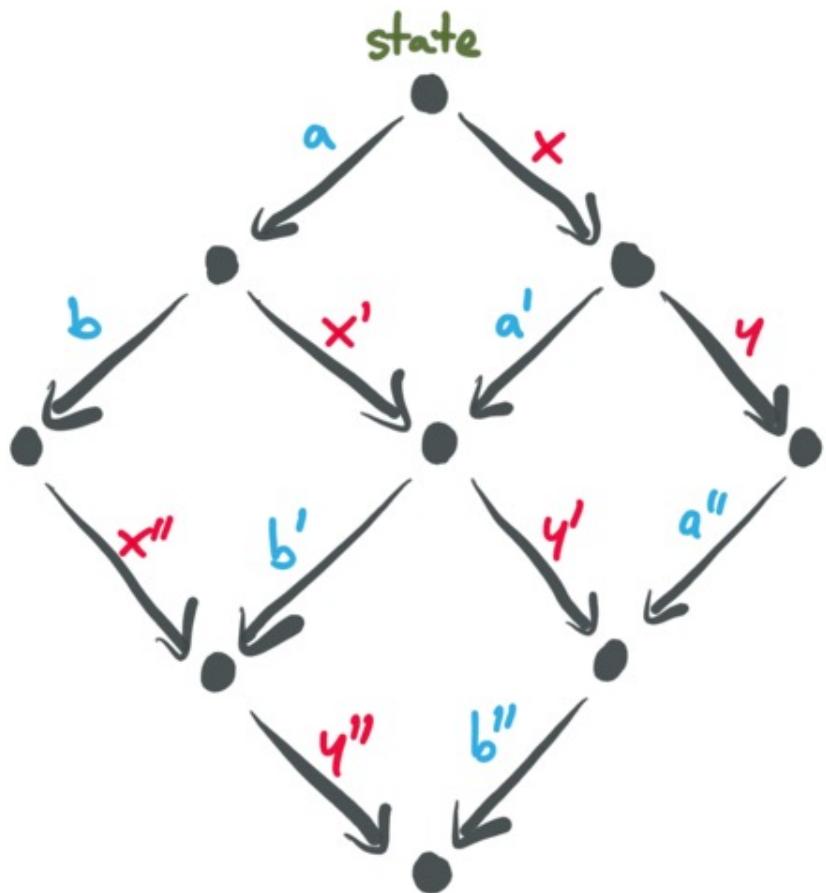


$$xform(a, x) = (a', x')$$

$$xform(b, x') = (b', x'')$$

$$xform(a', y) = (a'', y')$$

Diverged by more than one op?



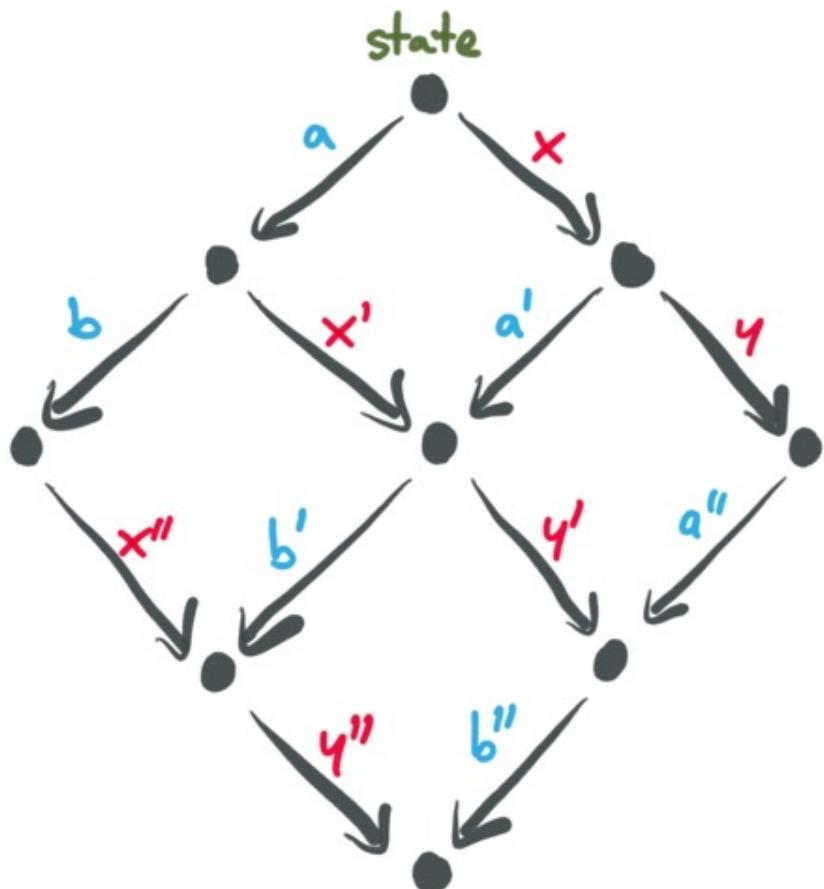
$$xform(a, x) = (a', x')$$

$$xform(b, x') = (b', x'')$$

$$xform(a', y) = (a'', y')$$

$$xform(b', y') = (b'', y'')$$

Diverged by more than one op?



$$\begin{aligned}xform(a, x) &= (a', x') \\xform(b, x') &= (b', x'') \\xform(a', y) &= (a'', y') \\xform(b', y') &= (b'', y'')\end{aligned}$$

Then

$$\begin{aligned}y''(x''(b(a(state)))) &= \\b''(a''(y(x(state))))\end{aligned}$$

Client and server each maintain a queue of operations they have sent to the other party, and that have not yet been acknowledged.
Transform each received op w.r.t. the ops in that queue.

Supporting multiple clients

- Independent instance of the protocol between the server and each client.

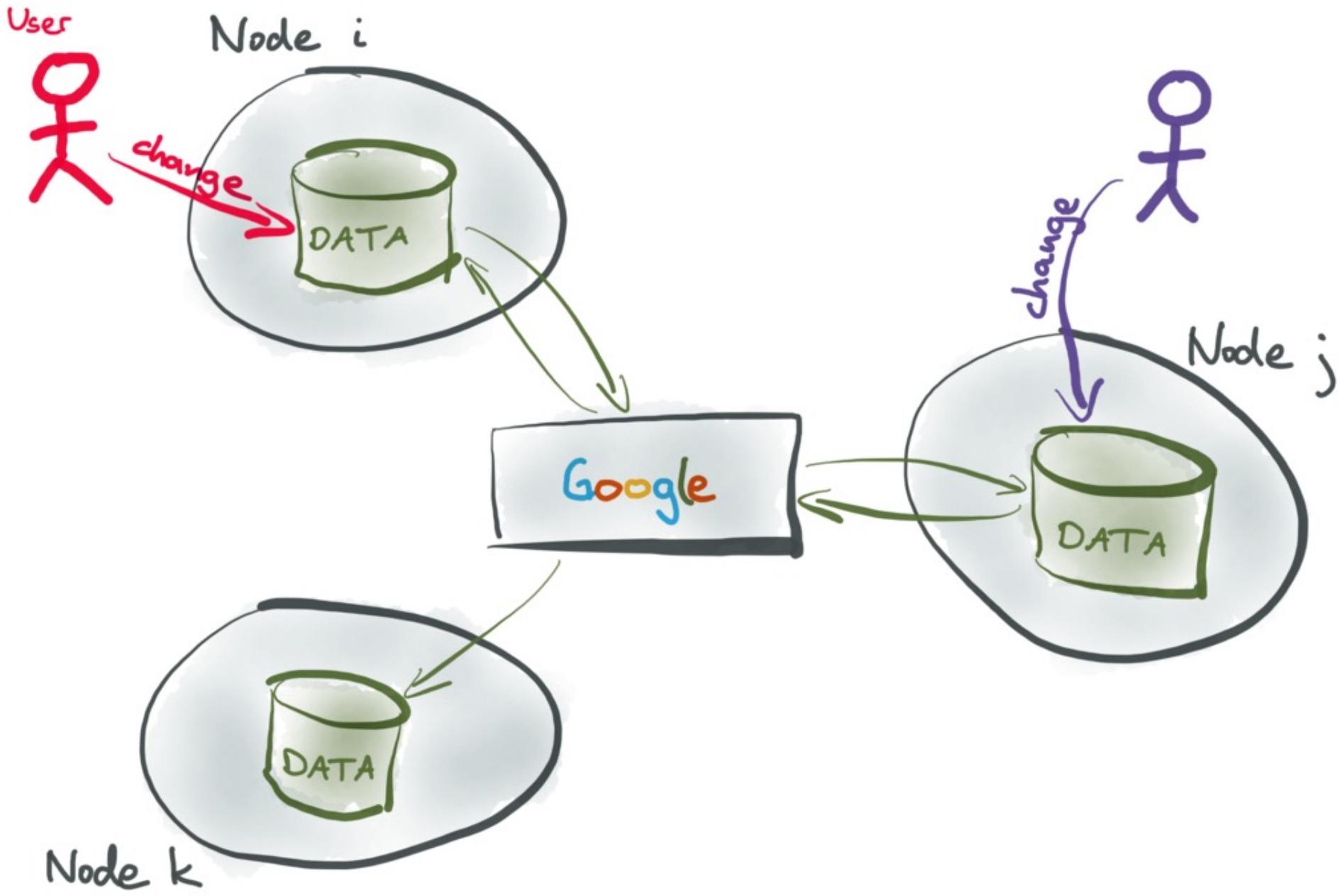
Supporting multiple clients

- Independent instance of the protocol between the server and each client.
- When an operation from one client has been transformed and applied to the server's state, it becomes a server operation from the other clients' point of view.

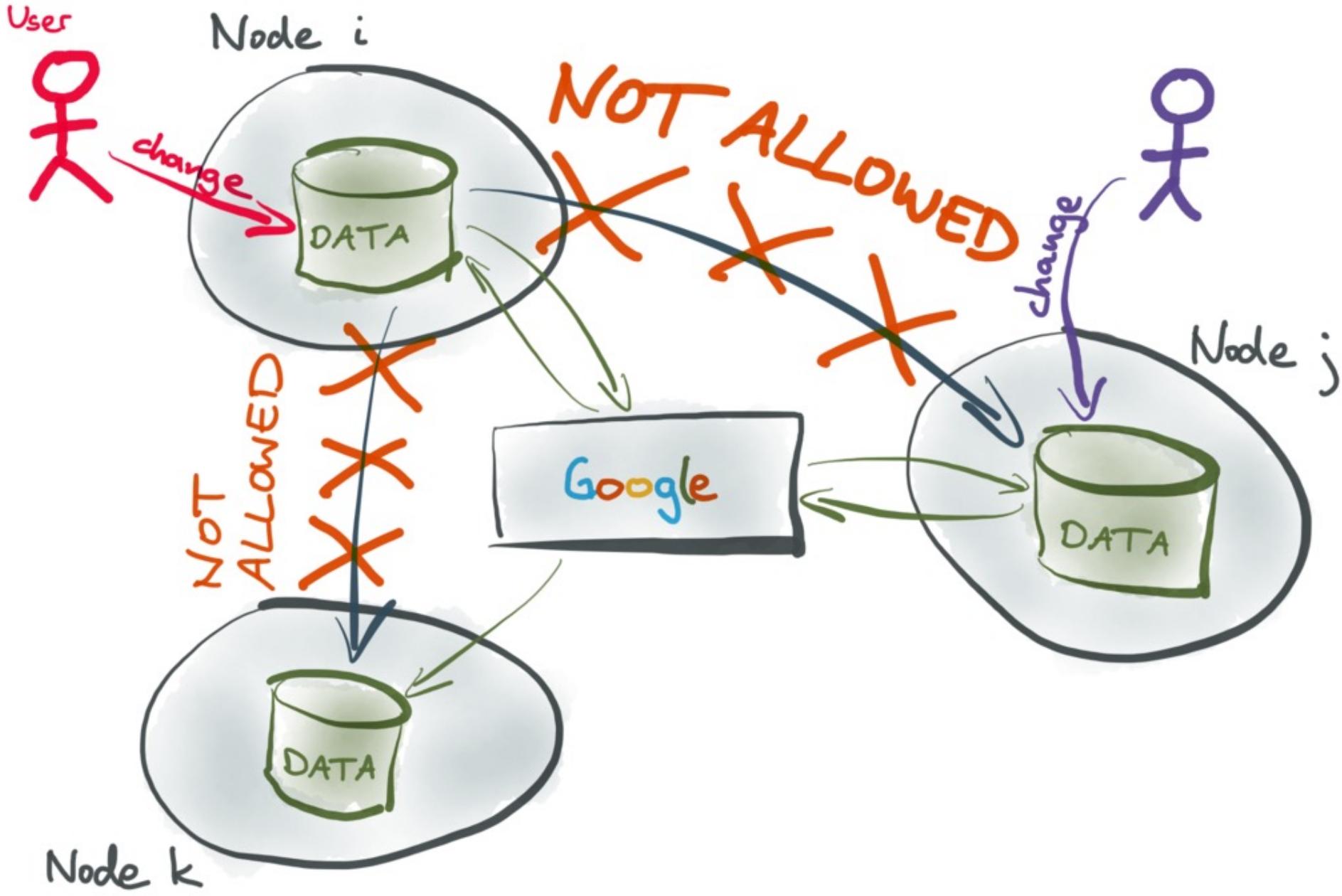
Supporting multiple clients

- Independent instance of the protocol between the server and each client.
- When an operation from one client has been transformed and applied to the server's state, it becomes a server operation from the other clients' point of view.
- The server has a single copy of the document for all clients, and the server applies operations sequentially.
- A server document version is identified by a single integer.

OPERATIONAL TRANSFORMATION IN GOOGLE DOCS.

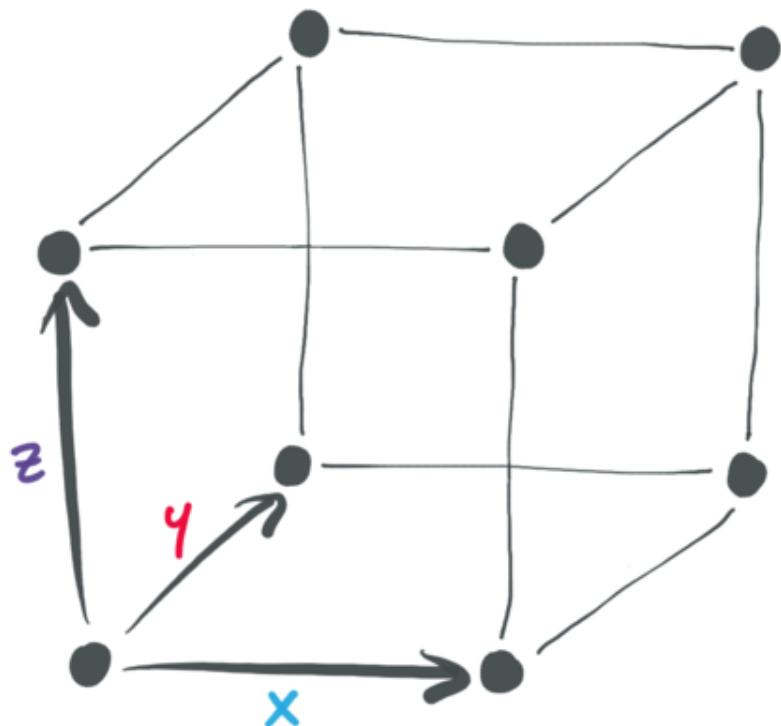


OPERATIONAL TRANSFORMATION IN GOOGLE DOCS.



Can we manage without a server?

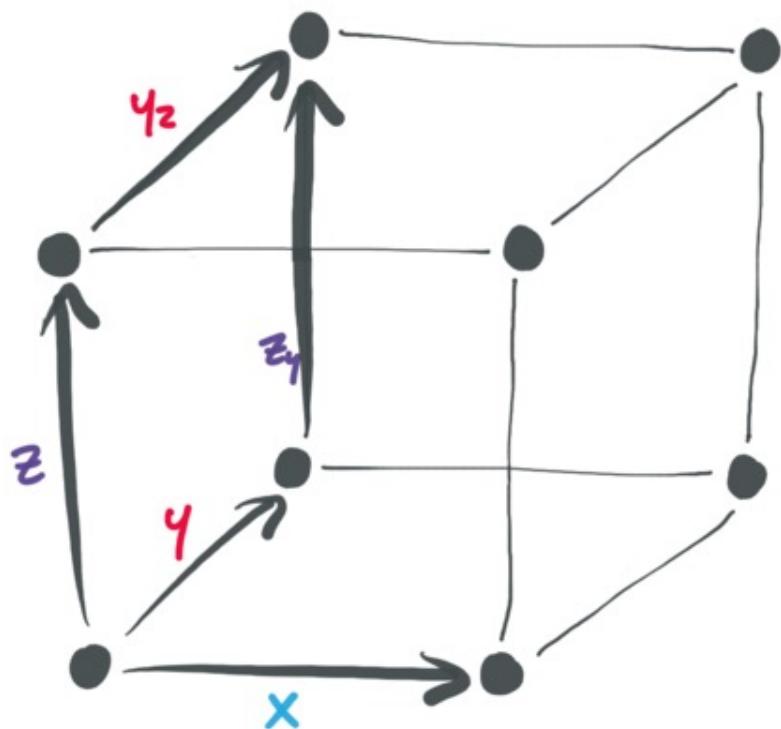
TP2 property:



Let x, y, z be concurrent ops
on the same state

Can we manage without a server?

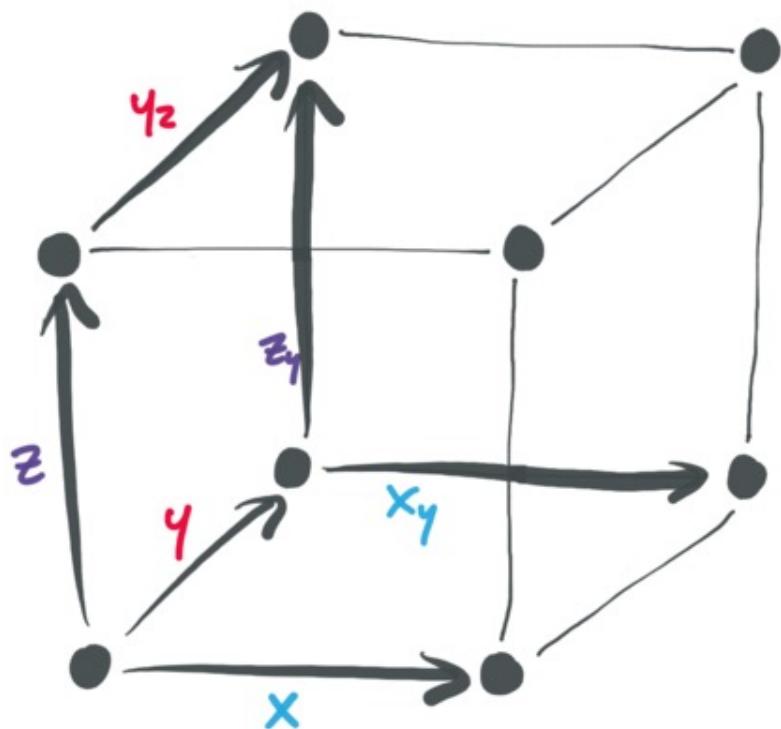
TP2 property:



Let x, y, z be concurrent ops on the same state, and let
 $\text{xform}(y, z) = (y_z, z_y)$

Can we manage without a server?

TP2 property:

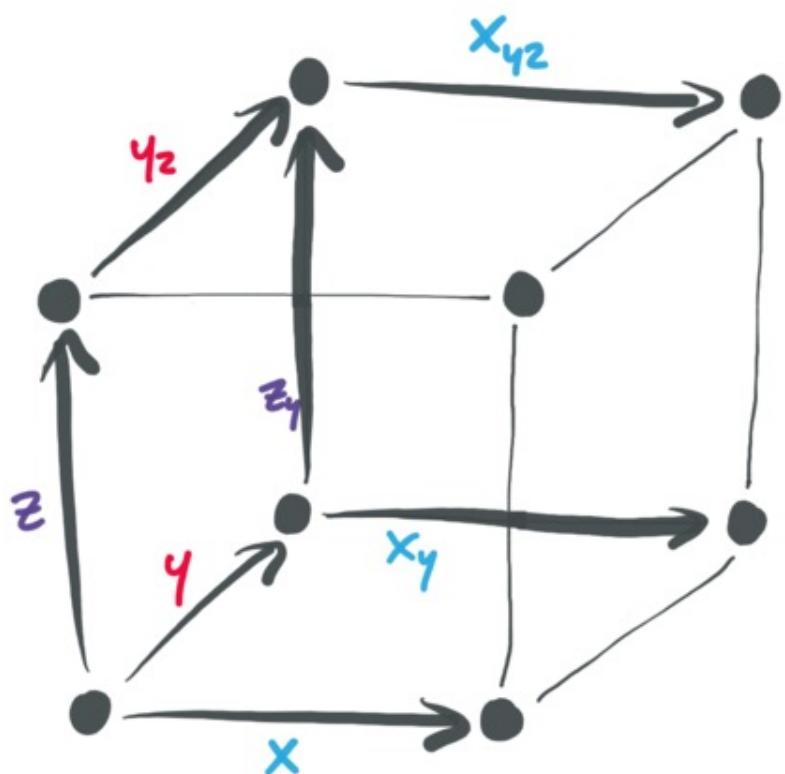


Let x, y, z be concurrent ops on the same state, and let

$$\text{xform}(y, z) = (y_z, z_y)$$

$$\text{xform}(x, y) = (x_y, -)$$

Can we manage without a server?



TP2 property:

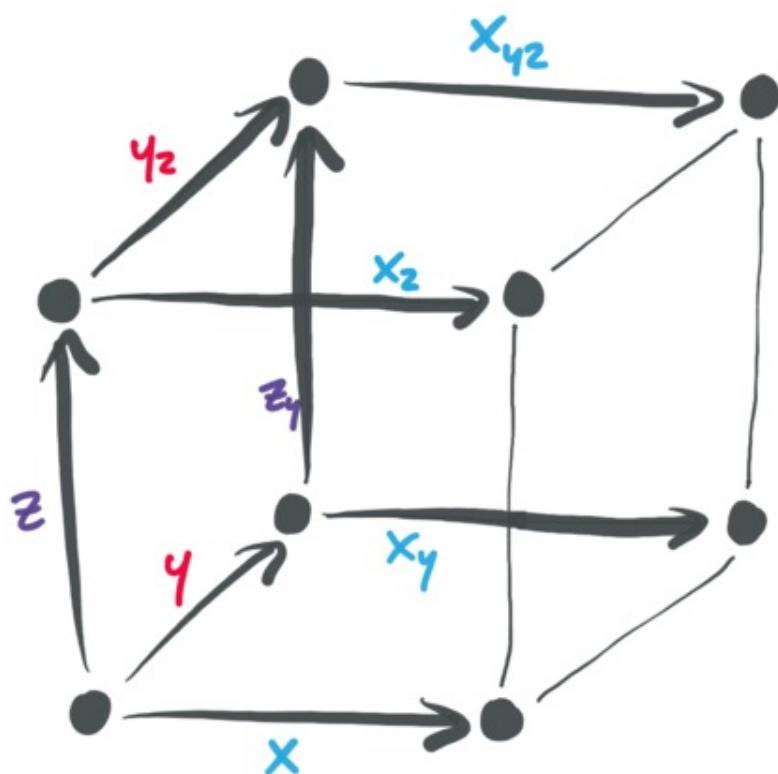
Let x, y, z be concurrent ops on the same state, and let

$$xform(y, z) = (y_z, z_y)$$

$$xform(x, y) = (x_y, -)$$

$$xform(x_y, z_y) = (x_{yz}, -)$$

Can we manage without a server?



TP2 property:

Let x, y, z be concurrent ops on the same state, and let

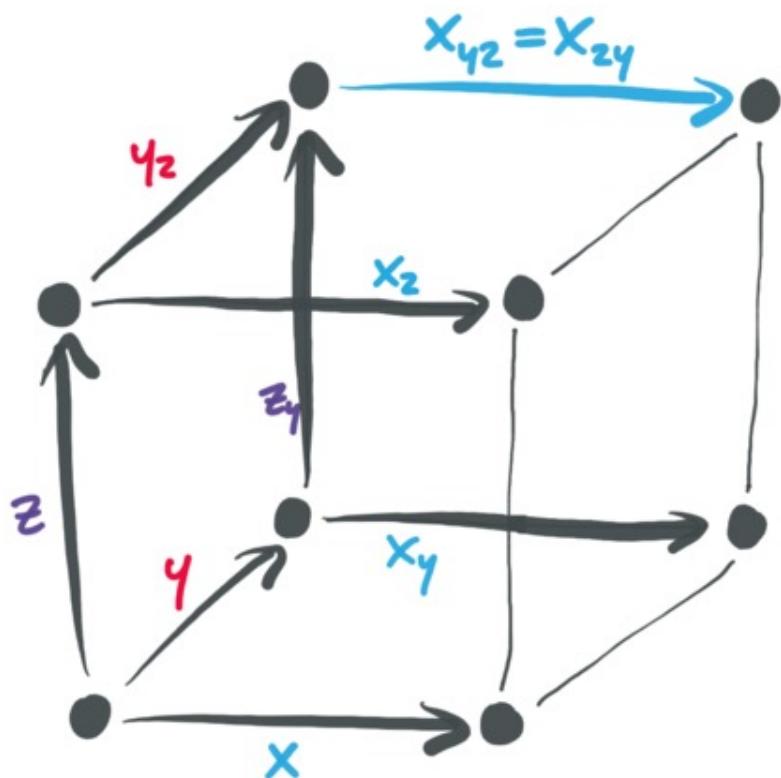
$$xform(y, z) = (y_z, z_y)$$

$$xform(x, y) = (x_y, -)$$

$$xform(x_y, z_y) = (x_{yz}, -)$$

$$xform(x, z) = (x_z, -)$$

Can we manage without a server?



TP2 property:

Let x, y, z be concurrent ops on the same state, and let

$$xform(y, z) = (y_z, z_y)$$

$$xform(x, y) = (x_y, -)$$

$$xform(x_y, z_y) = (x_{yz}, -)$$

$$xform(x, z) = (x_z, -)$$

$$xform(x_z, y_z) = (x_{zy}, -)$$

Then $x_{yz} = x_{zy}$.

FAILURES OF OPERATIONAL TRANSFORM

dOPT
Ellis & Gibbs
1989

adOPTed
Ressel et al.
1996

IMDR
Imine et al.
2003

Jupiter
Nichols et al.
1995

SOCT2
Suleiman et al.
1997

SDT
Li & Li
2004

SOCT 3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~adOPT~~
EU is ~~?~~ Gibbs
1989
WRONG

adOPTed
Ressel et al.
1996

IMDR
Imine et al.
2003

Jupiter
Nichols et al.
1995

SOCT2
Suleiman et al.
1997

SDT
Li & Li
2004

SOCT 3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~adOPT~~
EU is ~~?~~ Gibbs
1989
Wrong

~~adOPTed~~
Ressel et al.
1996
Wrong

IMDR
Imine et al.
2003

Jupiter
Nichols et al.
1995

SOCT2
Suleiman et al.
1997

SDT
Li & Li
2004

SOCT 3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~adOPT~~
EU is ~~?~~ Gibbs
1989
Wrong

~~adOPTed~~
Ressel et al.
1996
Wrong

~~IMDR~~
Imine et al.
2003

Jupiter
Nichols et al.
1995

~~SOCT2~~
Suleiman et al.
1997
Wrong

SDT
Li & Li
2004

SOCT 3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~adOPT~~
EU is Σ -Gibbs
1989
Wrong

~~adOPTed~~
Ressel et al.
1996
Wrong

~~IMDR~~
Imine et al.
2003
Wrong

Jupiter
Nichols et al.
1995

~~SDCT2~~
Suleiman et al.
1997
Wrong

SDT
Li & Li
2004

SDCT 3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~adOPT~~
EU is ²- Gibbs
1989
Wrong

~~adOPTed~~
Ressel et al.
1996
Wrong

~~IMDR~~
Imine et al.
2003
Wrong

Jupiter
Nichols et al.
1995

~~SDCT2~~
Suleiman et al.
1997
Wrong

~~SDT~~
Li & Li
2004
Wrong

SDCT 3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~adOPT~~
EU is 2. Gibbs
1989
Wrong

~~adOPTed~~
Ressel et al.
1996
Wrong

~~IMDR~~
Imine et al.
2003
Wrong

Jupiter
Nichols et al.
1995
~~require central server~~

~~SDCT 3/4~~
Vidot et al.
2000

~~SDCT~~
Suleiman et al.
1997
Wrong

~~SDT~~
Li & Li
2004
Wrong

TTF
Oster et al.
2006

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CROTs
3. Automerge
4. Rich text – Peritext
5. Non-interleaving – Fugue



Convergence guarantee :

Any two nodes have seen the
same set of operations

(but maybe in a different order!)



They are in the same state

Phone



sync

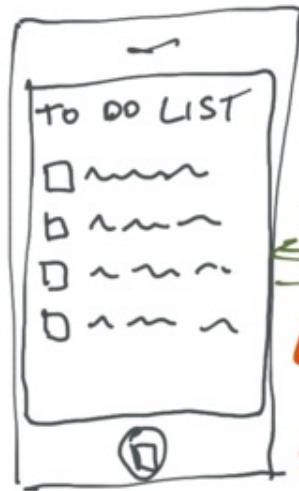


Laptop



sync

Phone



network
interruption

cloud!

≈ very high
network latency

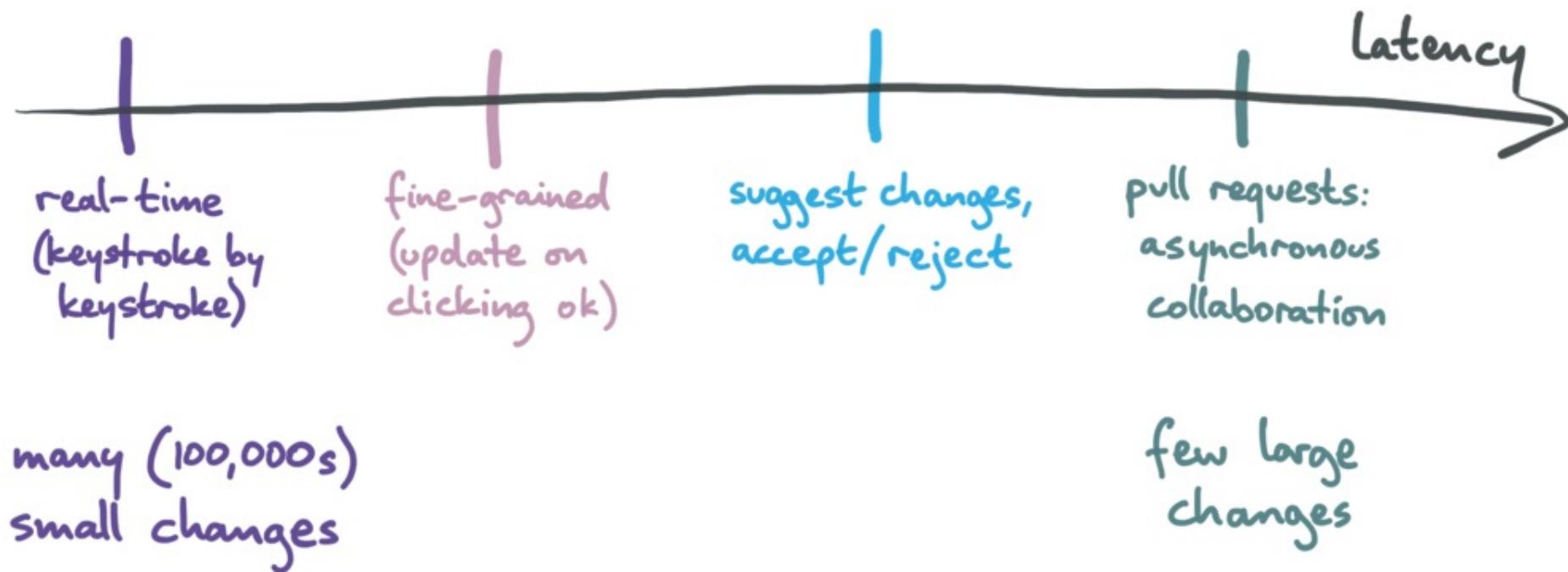
Laptop



Many situations require **working offline**: remote rural areas (agriculture, fieldwork), train in tunnel, on aeroplane, coffee shop wifi that cuts you off until you buy another drink, military, etc.

Collaboration

Syncing changes between several users





Share

Sign in

Normal text Arial 11 B I U A



Version control is not just for code

Use of version control systems (e.g. using Git) is ubiquitous among software developers. However, VCSes don't work well for "binary" files, such as powerpoint decks, spreadsheets, word docs, or vector graphics files – even though they are important to many professions. They need version control for the same reasons as why software developers need it: reviewing what your colleague changed yesterday; tentatively proposing changes to ask others' opinion; or experimenting with a new approach without disrupting others working on other parts of the file.

Microsoft Word's and Google Docs's suggestion mode are not really version control: they are visually messy, they offer no way of making a private branch, and it's difficult to review what has changed after a change has been accepted. Without version control, people use files named "Article revised v5 final (MK edits) FINAL.docx".

Upwelling

Combining real-time collaboration with version control for writers.

Collaboration features of existing software do not sufficiently meet the needs of professional non-fiction writers. Real-time collaboration in cloud applications like Google Docs can create stress when writers feel watched by their co-authors, and file-based collaboration can introduce difficulty with versioning and merging edits from different co-authors. Further, in environments where accuracy is crucial, like newsrooms, existing tools make reviewing changes unnecessarily difficult.

In the Upwelling project we have built an experimental editor that aims to satisfy the needs of professional writers and editors. It allows co-authors to collaborate in real time when they wish to, but it also supports work on private drafts that can be shared and merged only when their authors are ready. By combining elements of real-time collaboration with ideas from version control systems, Upwelling supports writers in maintaining their creative privacy and editors in ensuring accurate results.



Karissa Rae McKelvey

Scott Jenson

Eileen Wagner

Blaine Cook

Martin Kleppmann

March 2023



(not in a draft) ▾



Show changes ▾



Upwelling demo doc

NEW DRAFT

Version control is not just for code

Use of version control systems (e.g. using Git) is ubiquitous among software developers. However, VCSes don't work well for "binary" files, such as powerpoint decks, spreadsheets, word docs, or vector graphics files – even though they are important to many professions. They need version control for the same reasons as why software developers need it: reviewing what your colleague changed yesterday; tentatively proposing changes to ask others' opinion; or experimenting with a new approach without disrupting others working on other parts of the file.

Microsoft Word's and Google Docs's suggestion mode are not really version control: they are visually messy, they offer no way of making a private branch, and it's difficult to review what has changed after a change has been accepted. Without version control, people use files named "Article revised v5 final (MK edits) FINAL.docx".



H	e	L	o
0.2	0.4	0.6	0.8

↓ edit

H	e	L	L	o	!
0.2	0.4	0.6	0.7	0.8	0.9

H	e	L	o
0.2	0.4	0.6	0.8

$\{(0.2, A, "H"), (0.4, A, "e"),$
 $(0.6, A, "L"), (0.8, A, "o")\}$

↓ edit

H	e	L	L	o	!
0.2	0.4	0.6	0.7	0.8	0.9

$\{(0.7, A, "L"), (0.9, B, "!")\}$

=

$\{(0.2, A, "H"), (0.4, A, "e"),$
 $(0.6, A, "L"), (0.7, A, "L"),$
 $(0.8, A, "o"), (0.9, B, "!")\}$

>Hello!

0.2	0.4	0.6	0.7	0.8	0.9

$\{(0.2, A, "H"), (0.4, A, "e"),$
 $(0.6, A, "("), (0.7, A, "("),$
 $(0.8, A, "o"), (0.9, B, "!")\}$

actually a path
through a tree...
a few dozen bytes?

a UUID...
16 bytes binary/
36 bytes hex?

the actual ASCII/
Unicode character:
1 byte

m	i	l	k	\n
0.21	0.32	0.46	0.66	0.91

m	i	l	k	\n
0.21	0.32	0.46	0.66	0.91

User A

m	i	l	k	\n	e	g	g	s	\n
0.21	0.32	0.46	0.66	0.70	0.74	0.79	0.83	0.86	0.91

m	i	l	k	\n
0.21	0.32	0.46	0.66	0.91

User A

m	i	l	k	\n	e	g	g	s	\n
0.21	0.32	0.46	0.66	0.70	0.74	0.79	0.83	0.86	0.91

User B

m	i	l	k	\n	b	r	e	a	d	\n
0.21	0.32	0.46	0.66	0.72	0.75	0.77	0.80	0.84	0.88	0.91

m	i	l	k	\n
0.21	0.32	0.46	0.66	0.91

User A

m	i	l	k	\n	e	g	g	s	\n
0.21	0.32	0.46	0.66	0.70	0.74	0.79	0.83	0.86	0.91

User B

m	i	l	k	\n	b	r	e	a	d	\n
0.21	0.32	0.46	0.66	0.72	0.75	0.77	0.80	0.84	0.88	0.91

merged:

m	i	l	k	\n	\n	b	r	g	e	g	a	s	\n		
0.21	0.32	0.46	0.66	0.70	0.72	0.74	0.75	0.77	0.79	0.80	0.83	0.84	0.86	0.88	0.91

ORDERED LIST CRDT (NUTSHELL)

NODE A:

H	e	l	o
0a	1a	2a	3a

NODE B:

H	e	L	o
0a	1a	2a	3a

ORDERED LIST CRDT (NUTSHELL)

NODE A:

H	e	l	o	
0a	1a	2a	3a	

edit ID: 4a

H	e	l	l	o	
0a	1a	2a	4a	3a	

NODE B:

H	e	L	o	
0a	1a	2a	3a	

edit ID: 4b

H	e	l	o	!	
0a	1a	2a	3a	4b	

ORDERED LIST CRDT (NUTSHELL)

NODE A:

H	e	l	o	
0a	1a	2a	3a	

edit ID: 4a

H	e	l	L	o
0a	1a	2a	4a	3a

insert "L"
with id 4a
after id 2a



NODE B:

H	e	L	o	
0a	1a	2a	3a	

edit ID: 4b

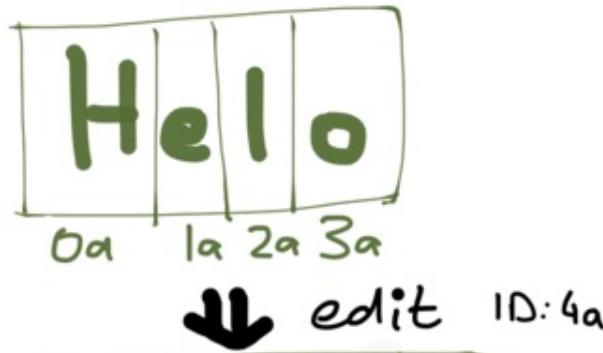
H	e	l	o	!
0a	1a	2a	3a	4b

insert "!" with
id 4b after id 3a



ORDERED LIST CRDT (NUTSHELL)

NODE A:



insert "L"
with id 4a
after id 2a

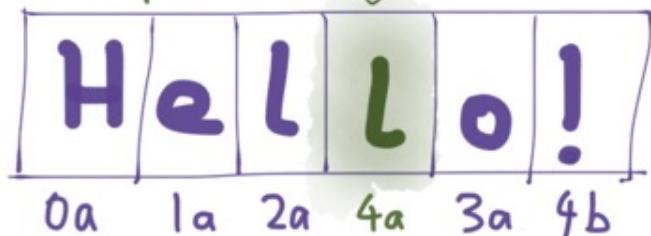


NODE B:



insert "!" with
id 4b after id 3a

insert "L" with
id 4a after id 2a



ORDERED LIST CRDT (NUTSHELL)

NODE A:

H	e	l	o	
0a	1a	2a	3a	

edit ID: 4a

H	e	l	l	o	
0a	1a	2a	4a	3a	

insert "L"
with id 4a
after id 2a



✓ insert "!" with
id 4b after id 3a

H	e	l	L	l	o	!	
0a	1a	2a	4a	3a	4b		

NODE B:

H	e	L	o	
0a	1a	2a	3a	

edit ID: 4b

H	e	l	o	!	
0a	1a	2a	3a	4b	

insert "!" with
id 4b after id 3a

✓ insert "L" with
id 4a after id 2a

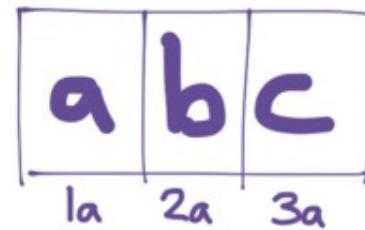
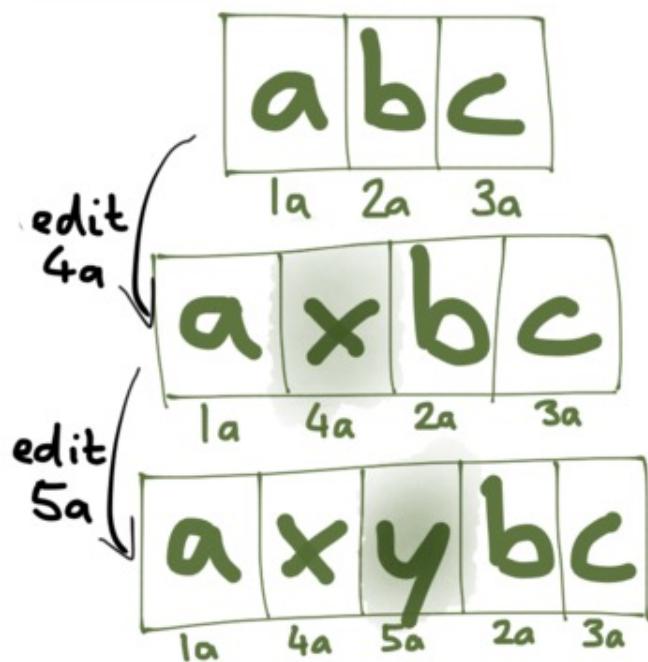
H	e	L	l	o	!	
0a	1a	2a	4a	3a	4b	

INSERTING IN THE SAME PLACE

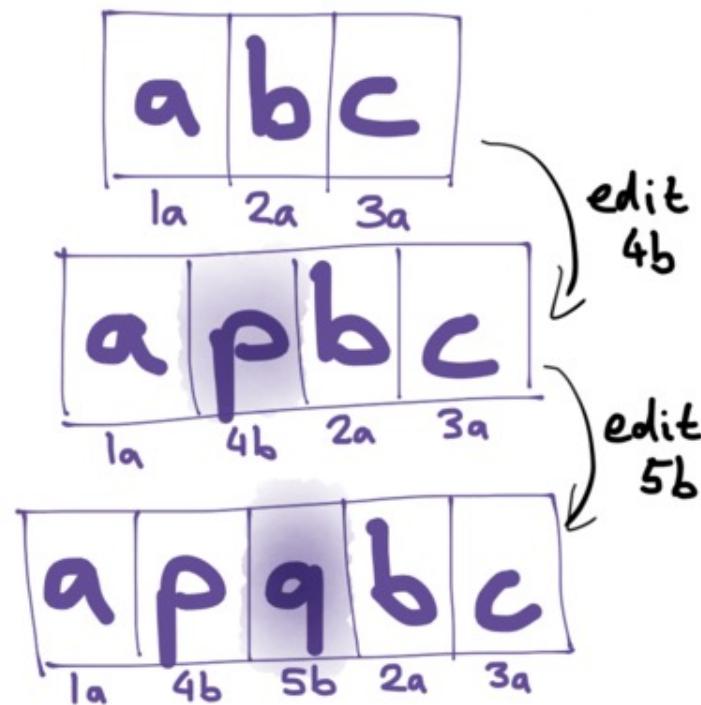
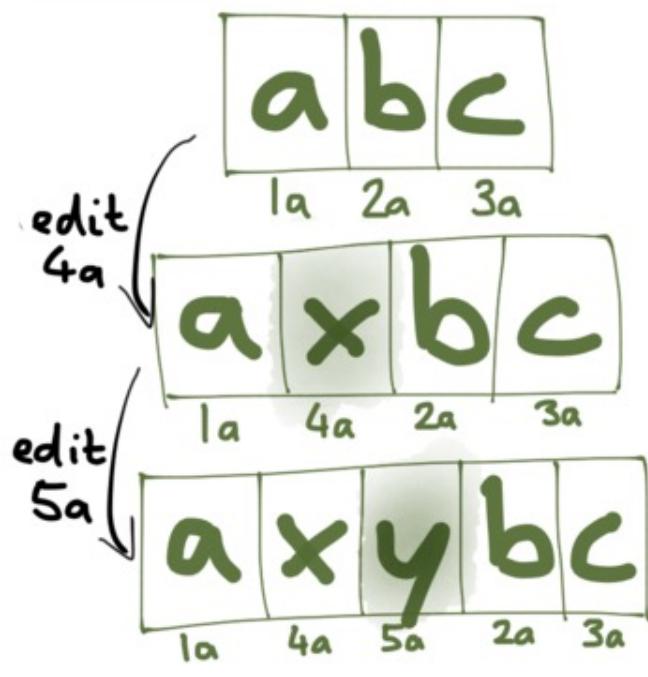
a	b	c
1a	2a	3a

a	b	c
1a	2a	3a

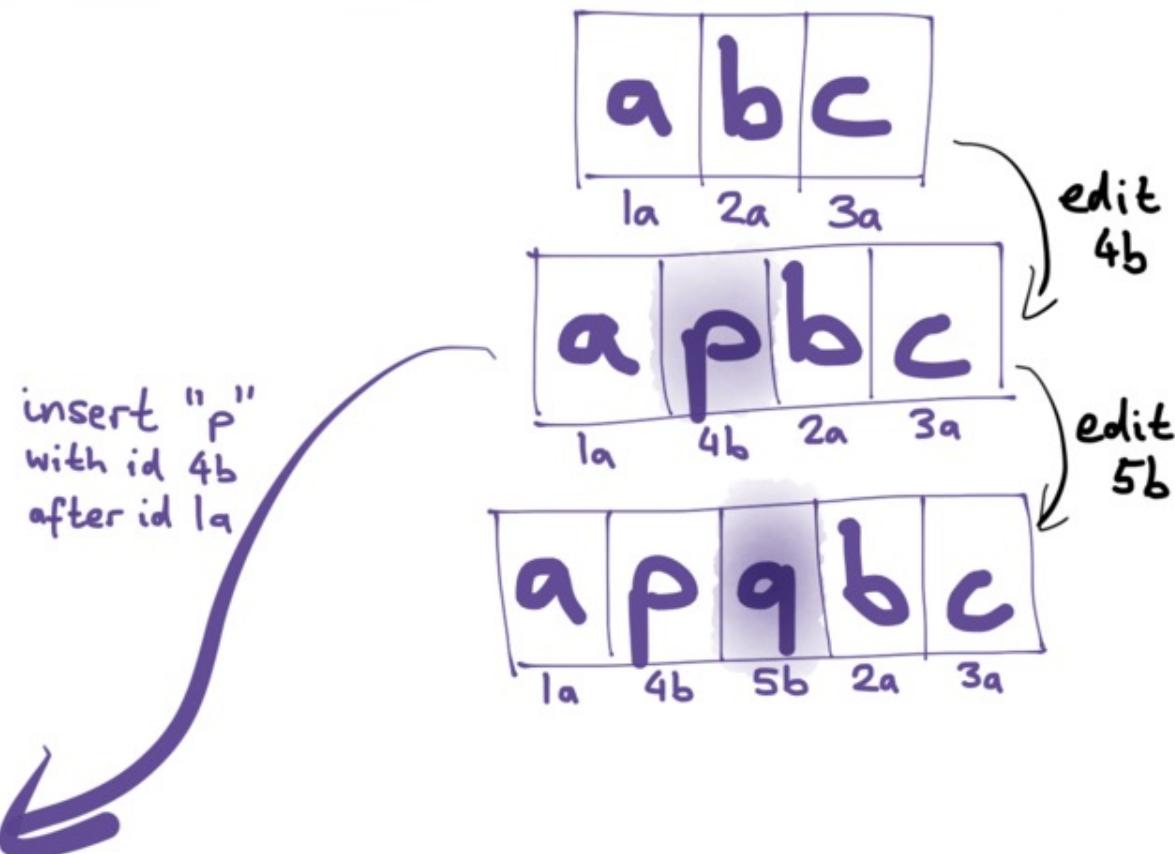
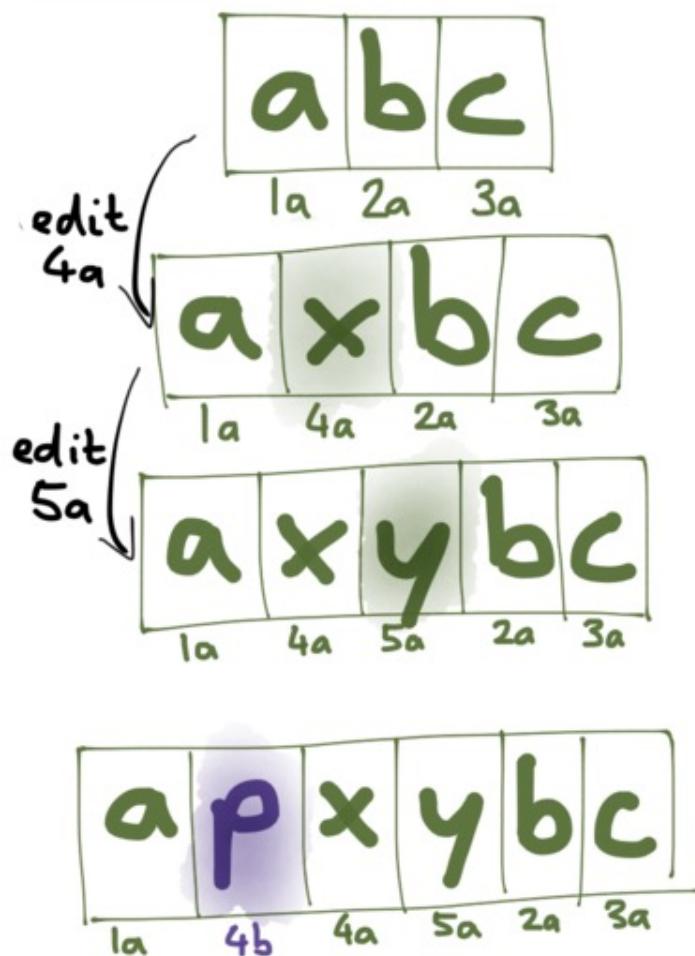
INSERTING IN THE SAME PLACE



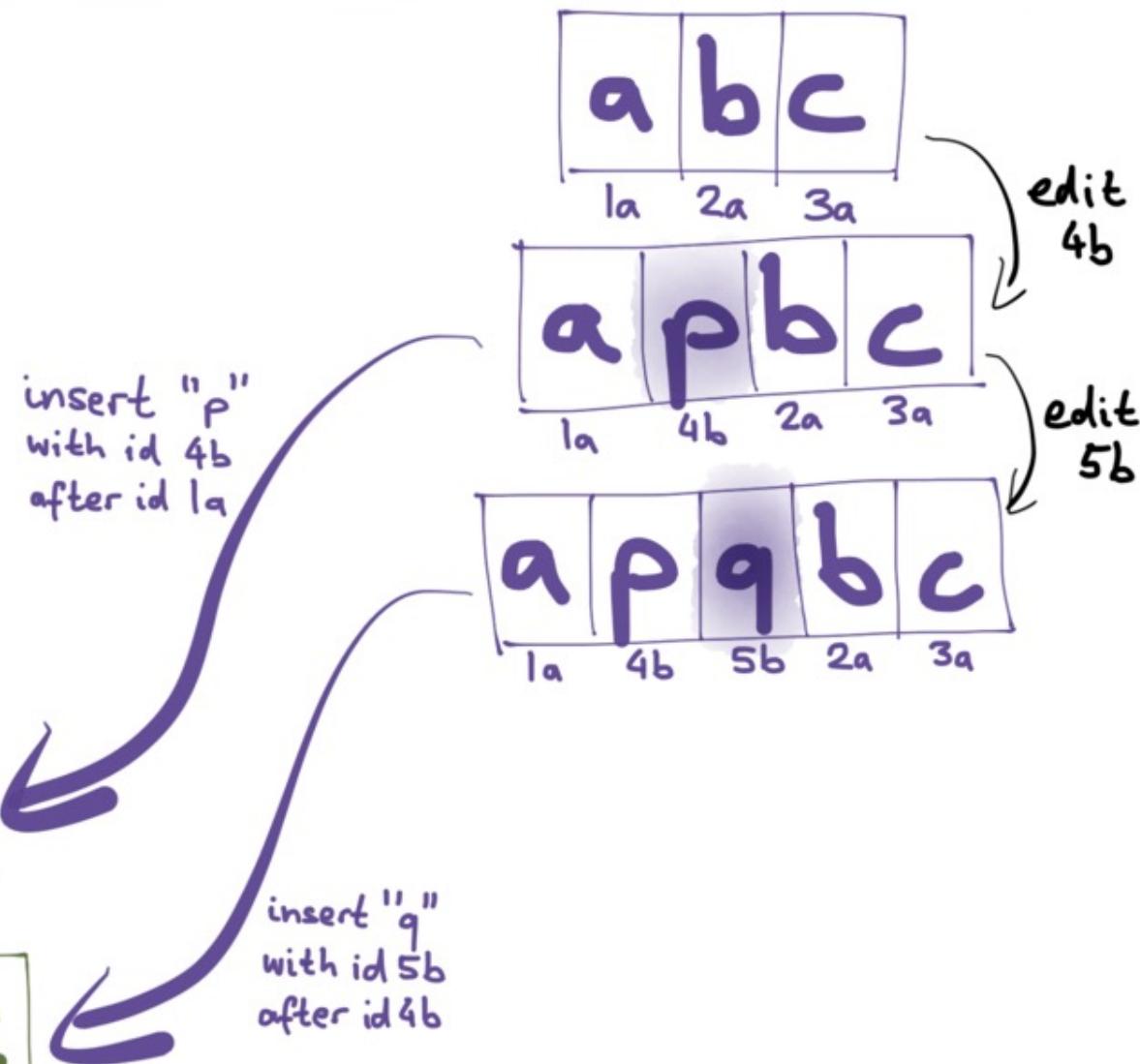
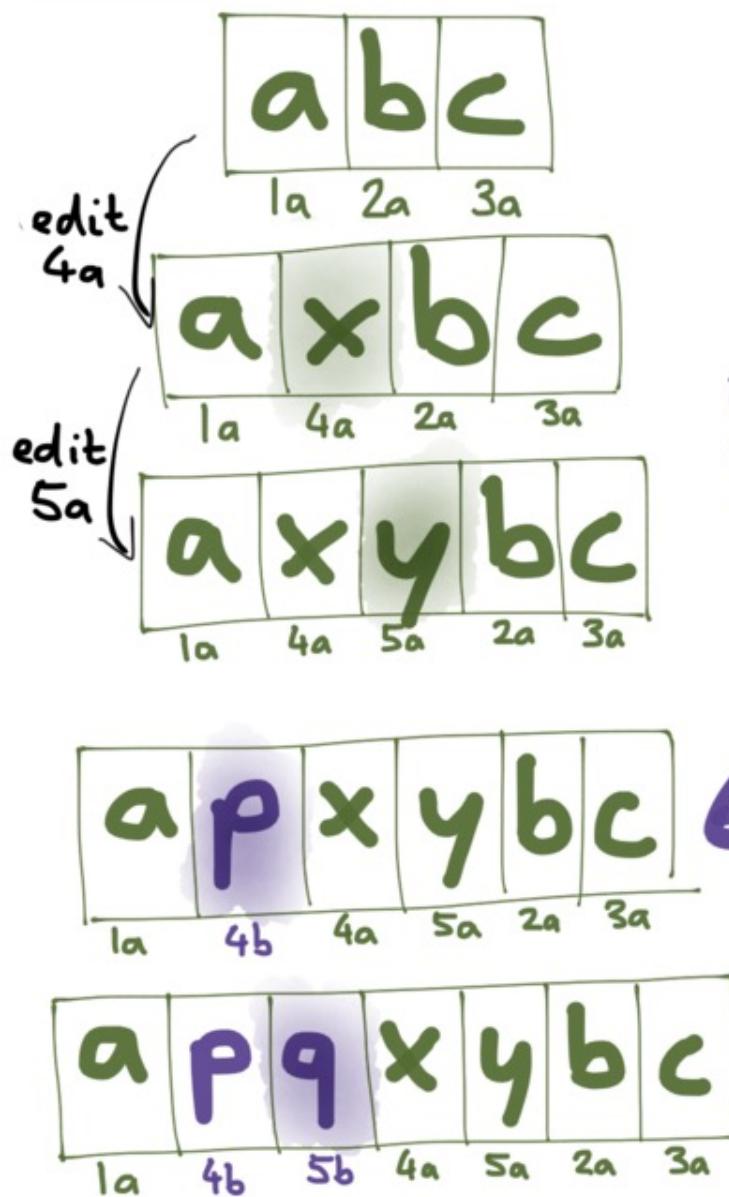
INSERTING IN THE SAME PLACE



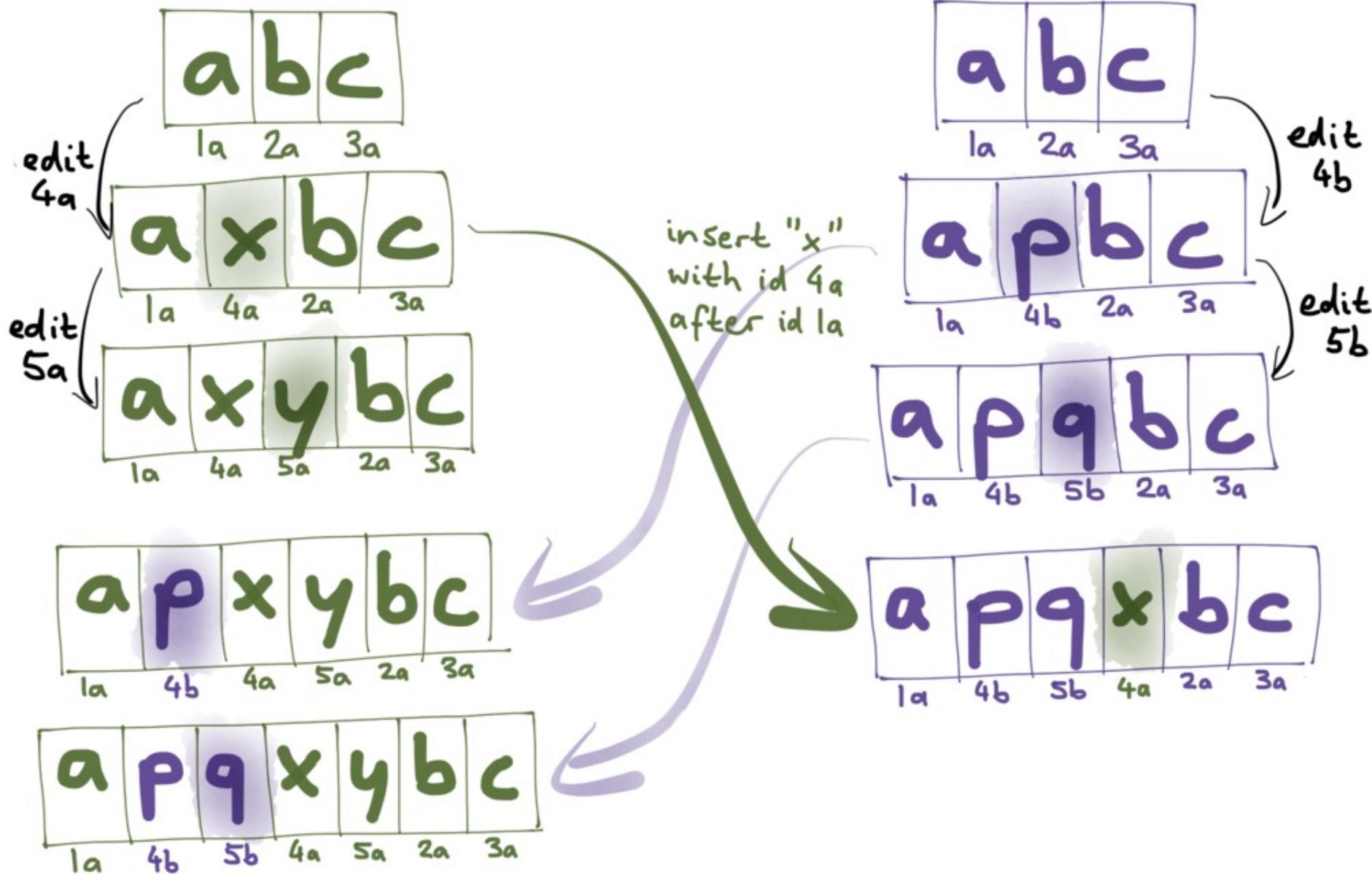
INSERTING IN THE SAME PLACE



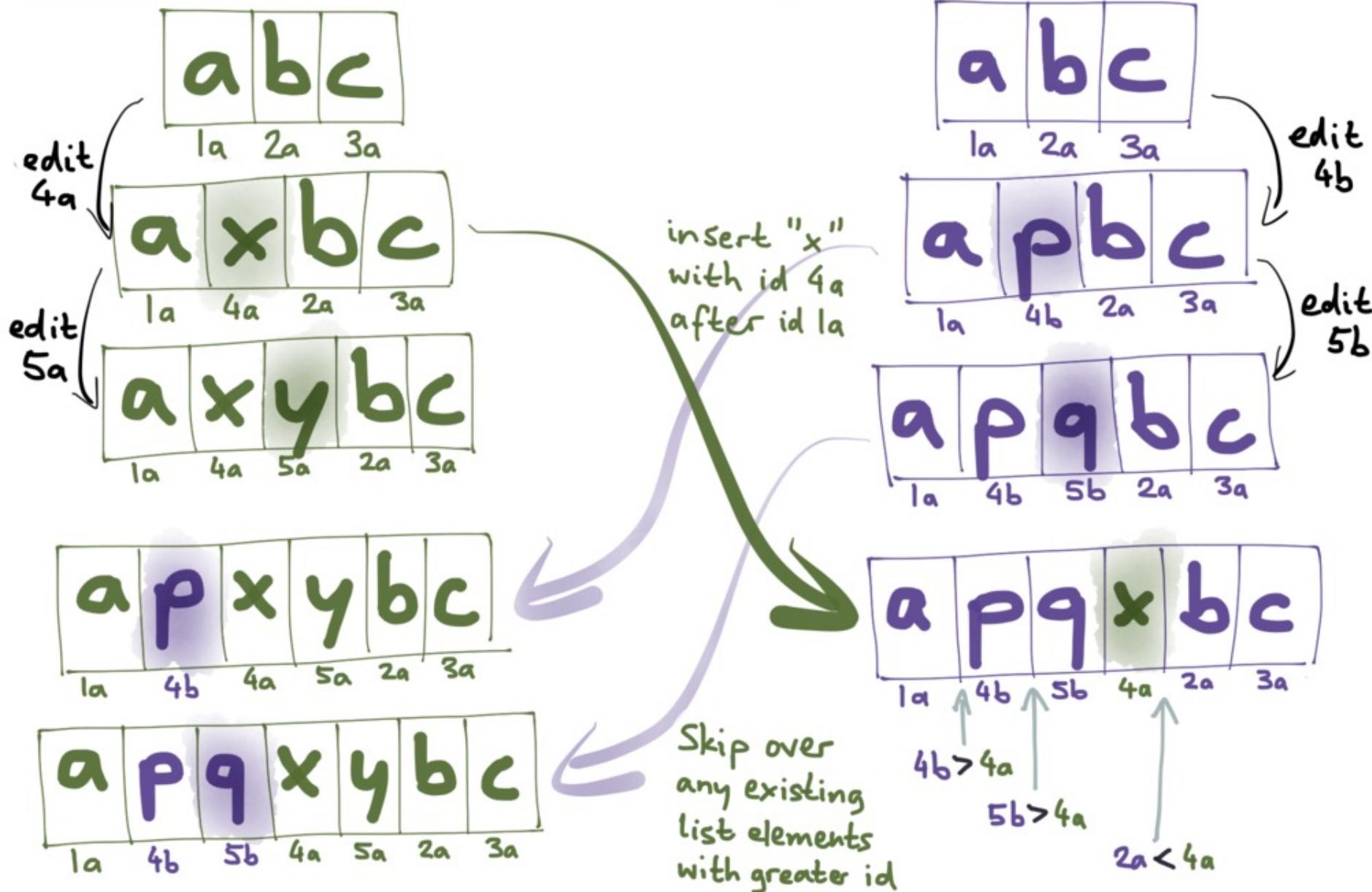
INSERTING IN THE SAME PLACE



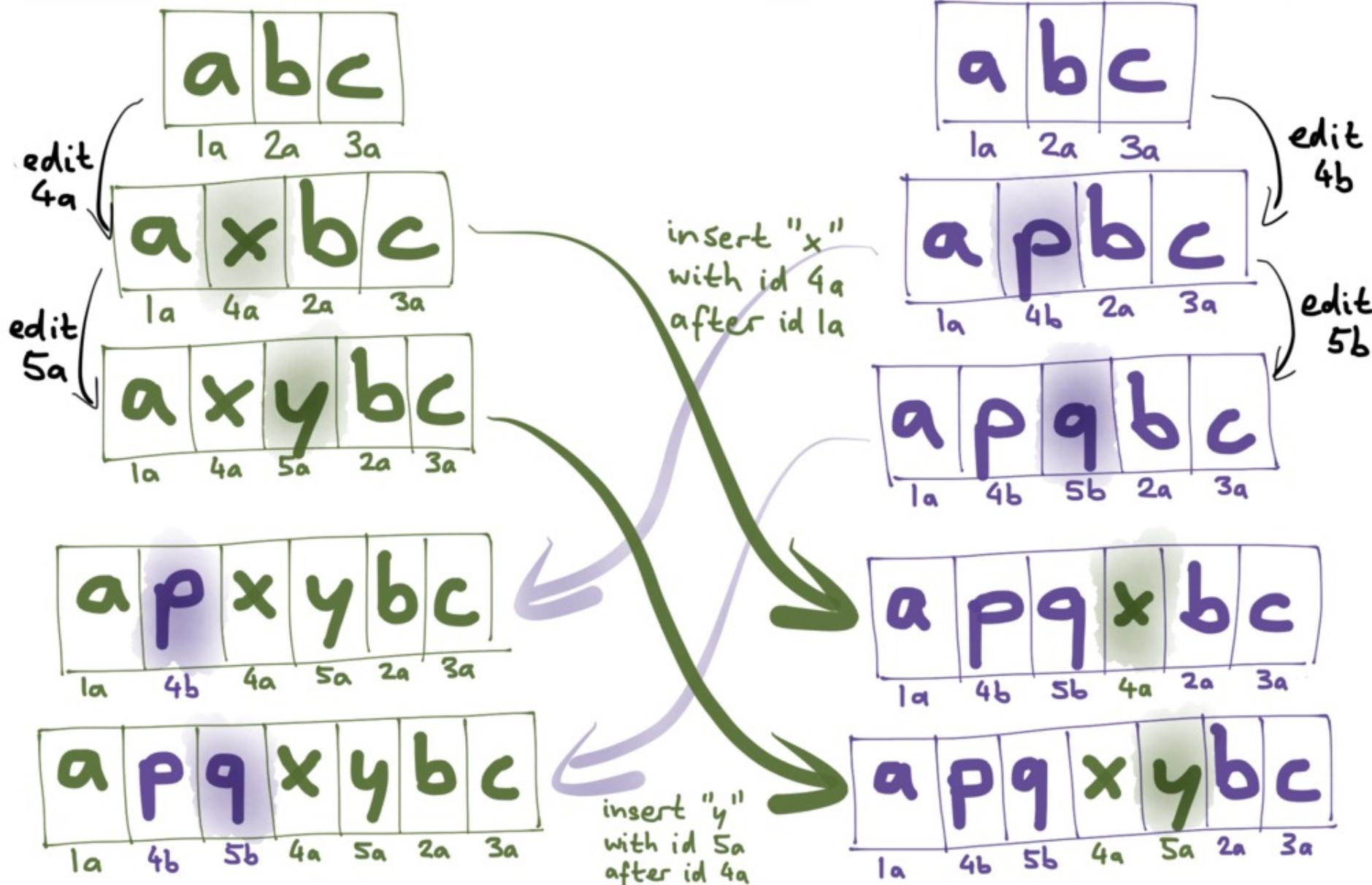
INSERTING IN THE SAME PLACE



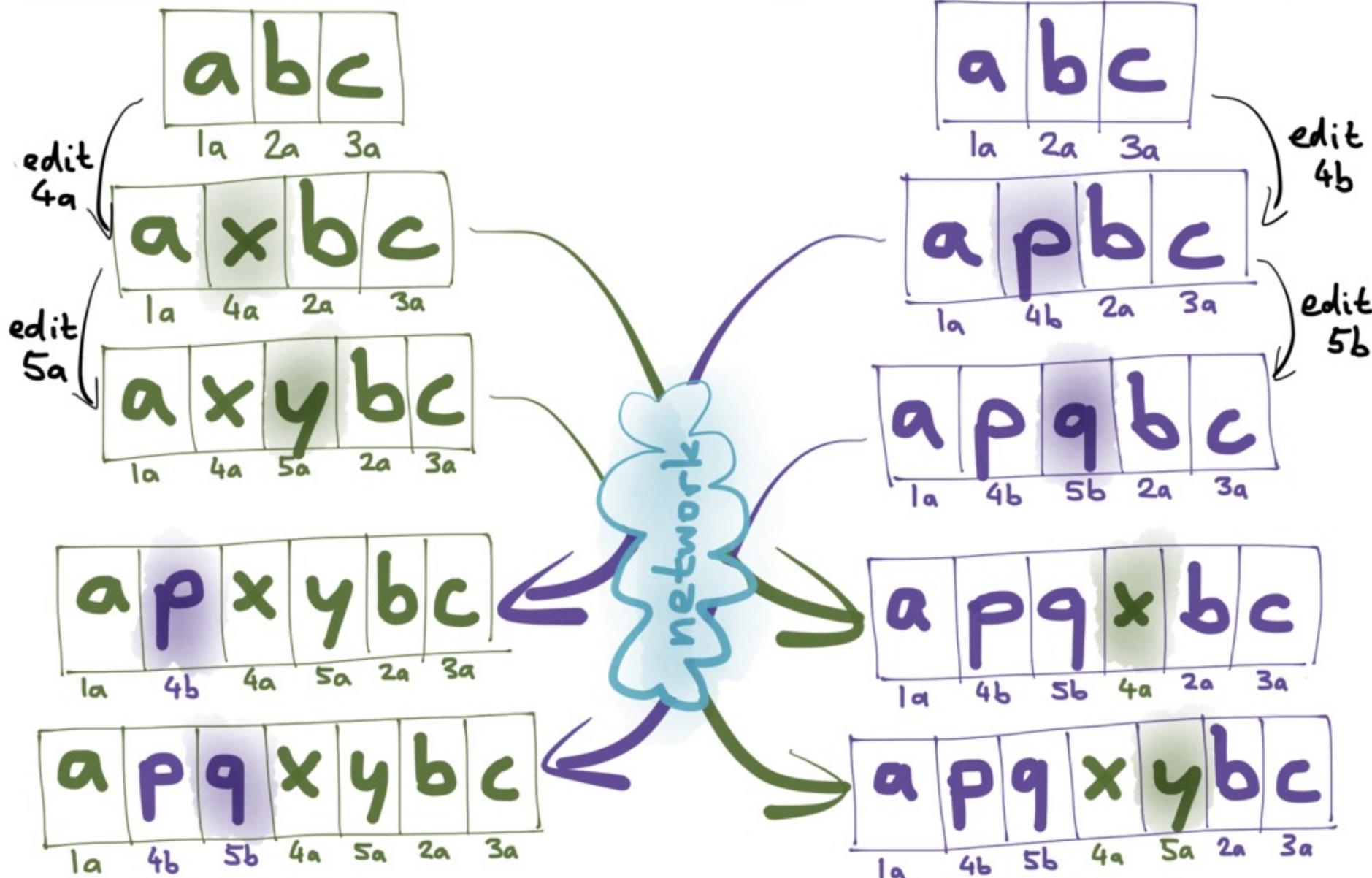
INSERTING IN THE SAME PLACE



INSERTING IN THE SAME PLACE



INSERTING IN THE SAME PLACE

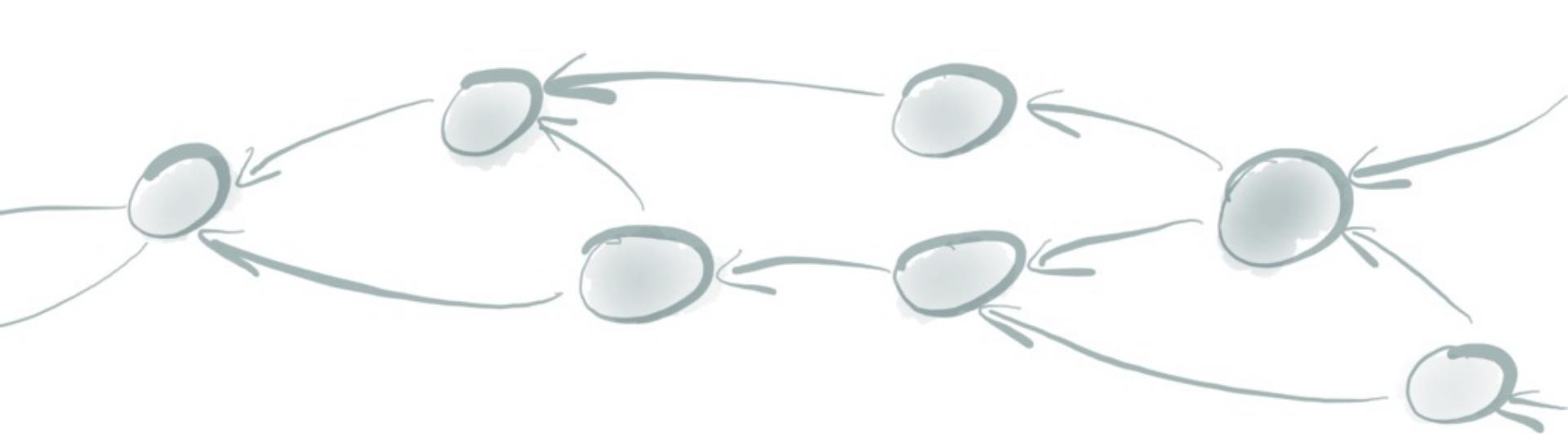


	OT	CRDT
Real-time collaboration		
Offline support		
Branching/merging		
Arbitrary JSON		
Rich text		
Edit history		
Metadata overhead		
Decentralisation		

	OT	CRDT
Real-time collaboration		
Offline support		
Branching/merging		
Arbitrary JSON		
Rich text		
Edit history		 Automerge turning these into
Metadata overhead		
Decentralisation		

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text – Peritext
5. Non-interleaving – Fugue



Automerger

<https://github.com/automerge/automerge>

log of changes

insert
"H"

log of changes



log of changes

- 
- insert "H"
 - insert "e"
 - insert "L"

log of changes

-
- insert "H"
insert "e"
insert "L"
insert "L"

log of changes

insert
"H"

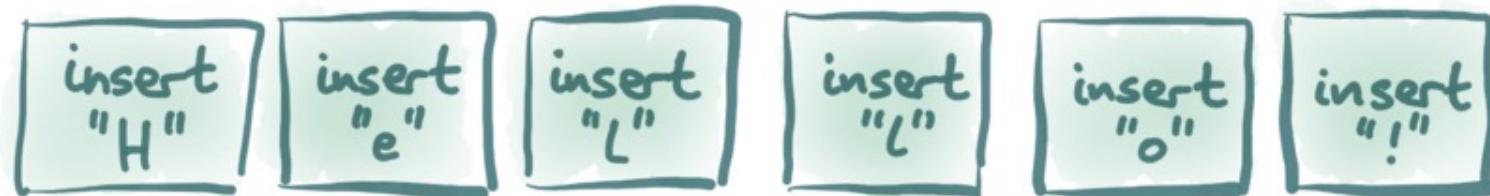
insert
"e"

insert
"L"

insert
"L"

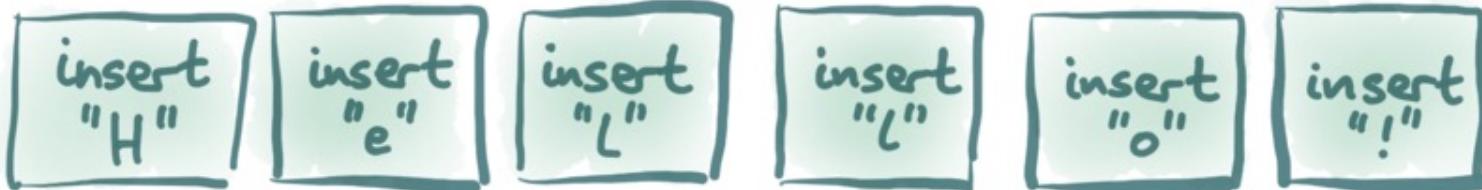
insert
"o"

log of changes



Each change
is encoded as
a byte array

log of changes

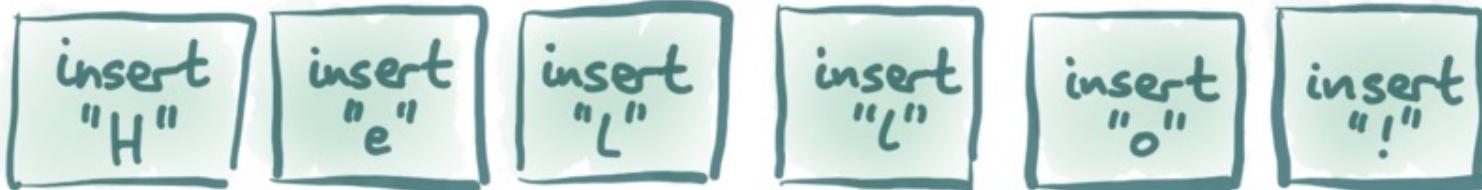


Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot

snapshot
"Hello!"

log of changes



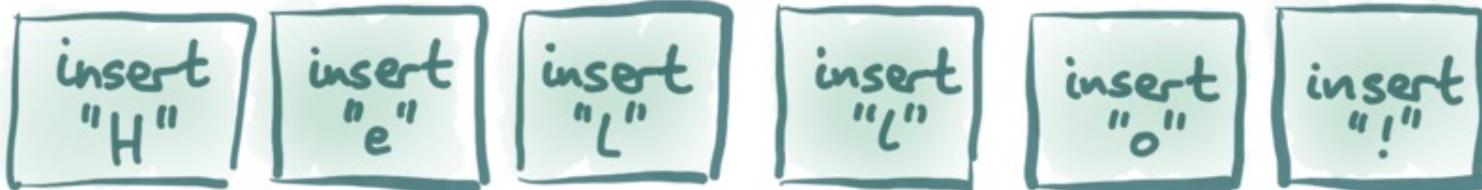
Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot

snapshot
"Hello!"

delete
"!"

log of changes



Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot

snapshot
"Hello!"

delete
"!"

insert
" "

log of changes



Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot

snapshot
"Hello!"

delete
"!"

insert
" "

insert
"w"

log of changes



Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot

snapshot
"Hello!"

delete
"!"

insert
" "

insert
"w"

insert
"o"

log of changes

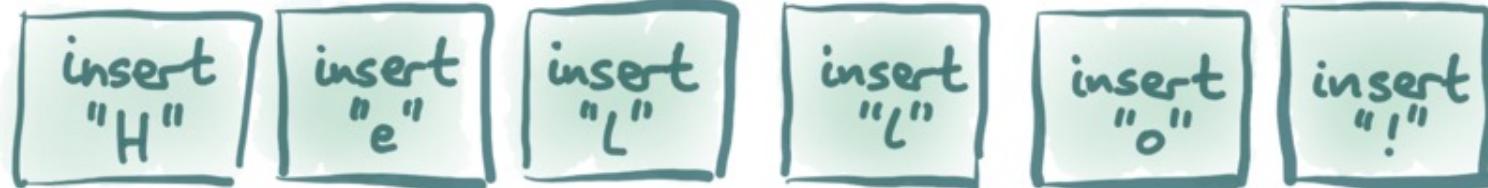


Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot



log of changes



Each change
is encoded as
a byte array

from time to time, compress
the log into a compact snapshot



Automerge compression benchmark

Test document: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper)

- 182,315 single-character insertions
- 77,463 single-character deletions

Final text (uncompressed, no edit history): 105 kB

Automerge compression benchmark

Test document: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper)

- 182,315 single-character insertions
- 77,463 single-character deletions

Final text (uncompressed, no edit history): 105 kB

Full edit history (columnar compression): 184 kB
(0.7 bytes/operation!)

Can look at any past document version, diffing, branching, merging ...

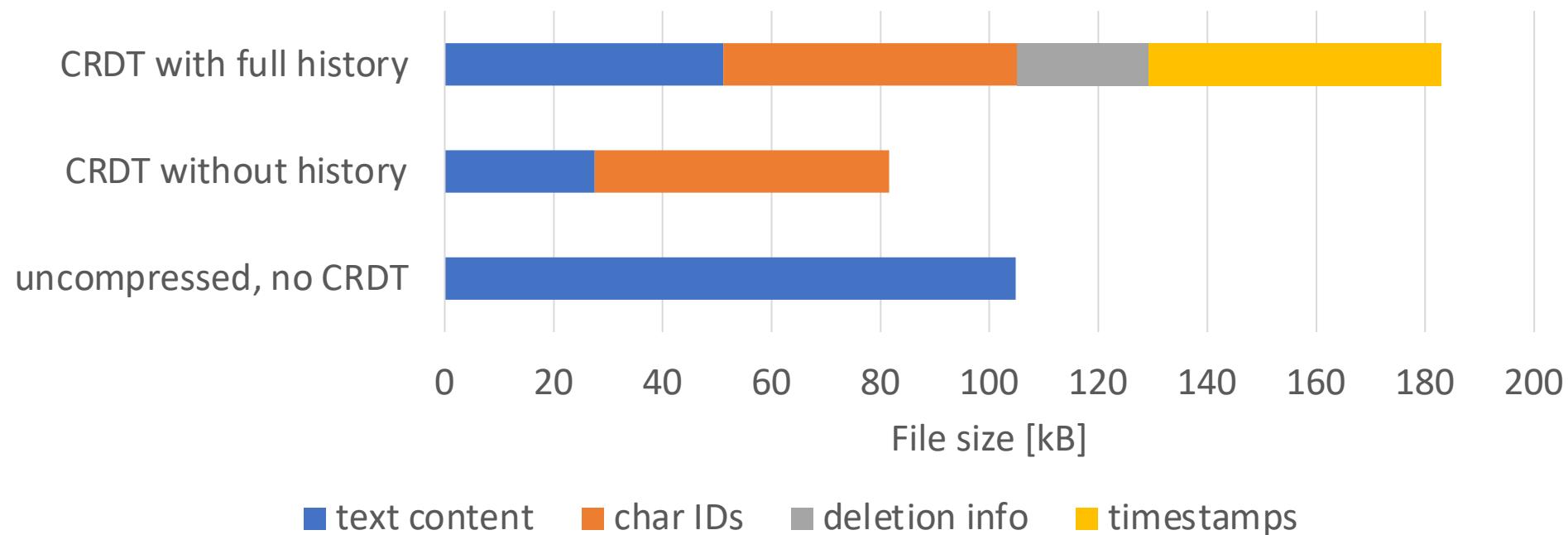
Automerge compression benchmark

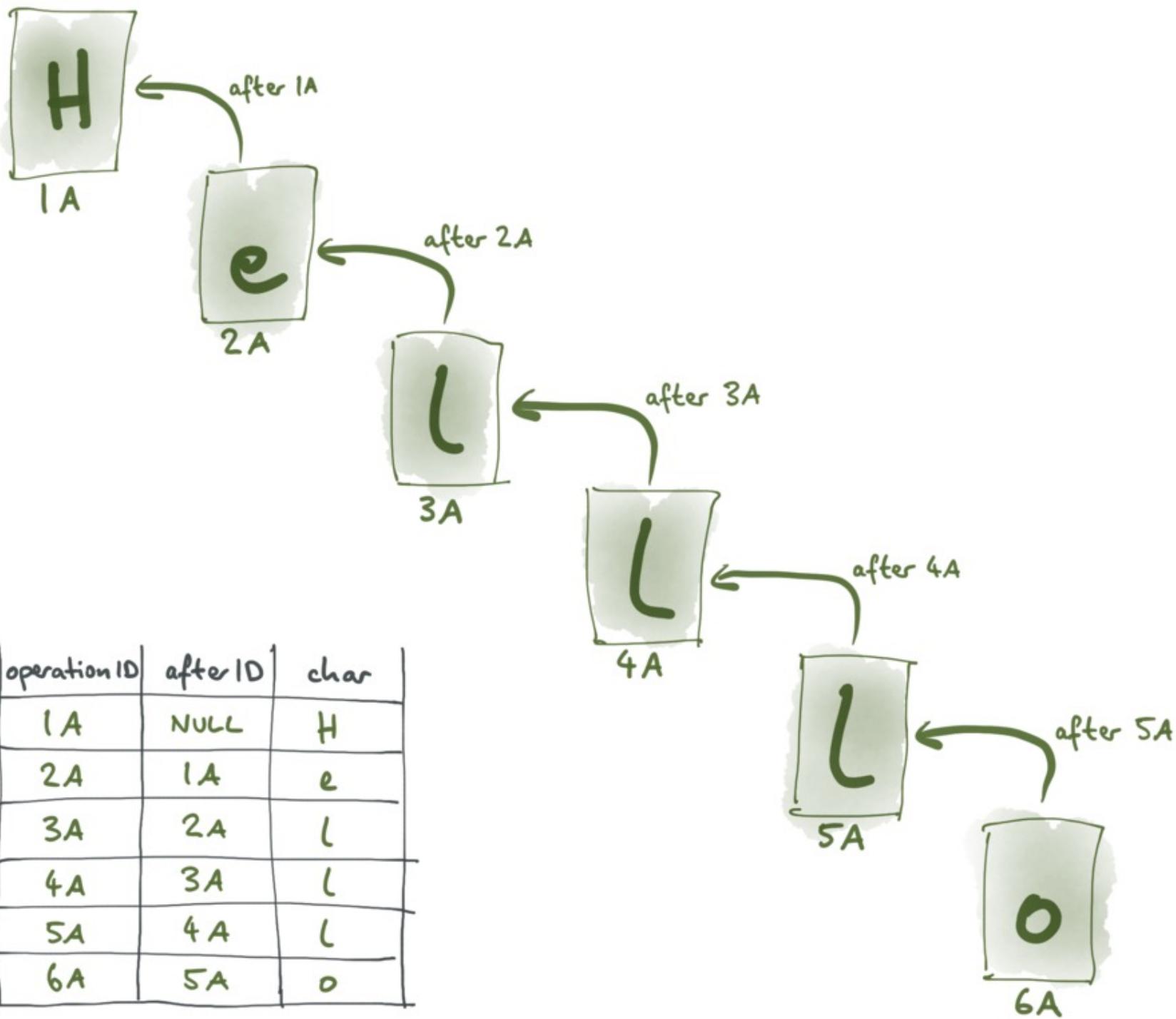
Benchmark data: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper) containing 182,315 single-character insertions and 77,463 single-character deletions, timestamped with 1-second granularity.

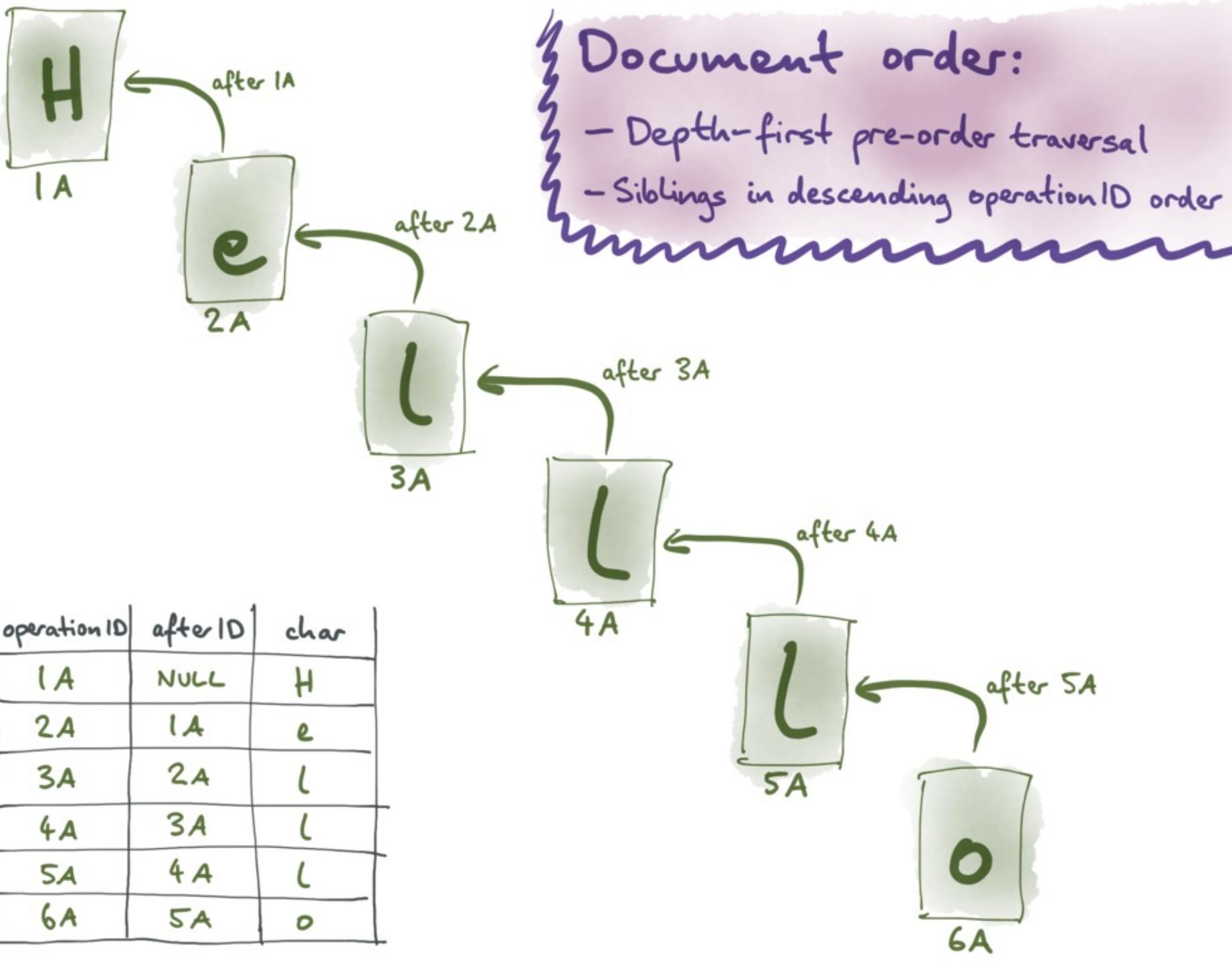
As individual changes: 33.7 MB (130 bytes/operation)

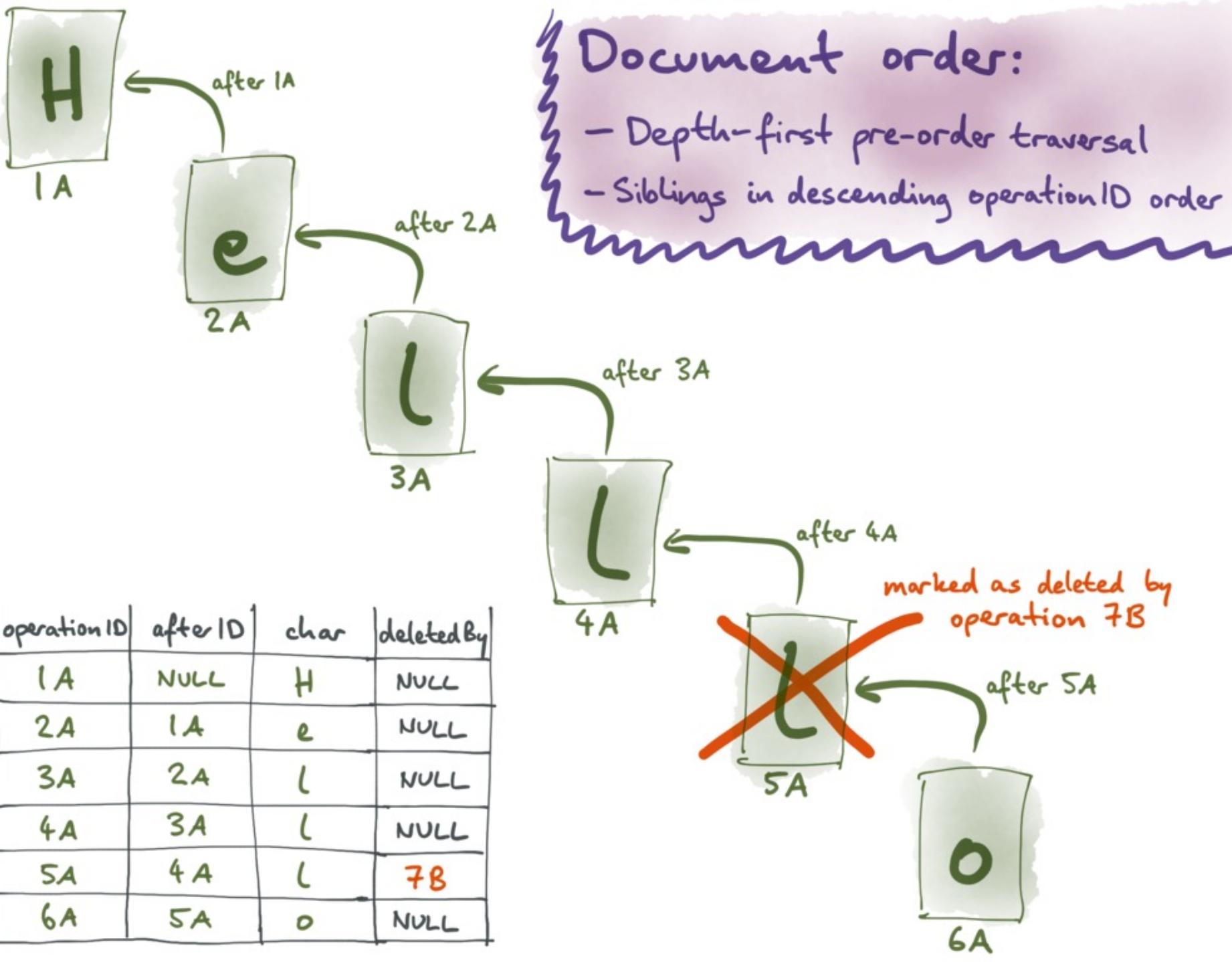
As compressed document with full edit history: 184 kB (0.7 bytes/operation)

Breakdown of compressed columnar file contents









COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	-	-	1	"H"	-	-
2	A	1	A	1	"e"	-	-
3	A	2	A	1	"l"	-	-
4	A	3	A	1	"l"	-	-
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	-	-

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	-	-	1	"H"	-	-
2	A	1	A	1	"e"	-	-
3	A	2	A	1	"("	-	-
4	A	3	A	1	")"	-	-
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	-	-

→ 1, 2, 3, 4, 5, 6

delta-encode to 1, 1, 1, 1, 1, 1

run-length encode to (6, 1)

LEB128 encodes this in 2 bytes

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	-	-	1	"H"	-	-
2	A	1	A	1	"e"	-	-
3	A	2	A	1	"("	-	-
4	A	3	A	1	")"	-	-
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	-	-

make a lookup table: $\{ "A": 0, "B": 1 \}$
 $\rightarrow 0, 0, 0, 0, 0, 0$

\rightarrow run-length encode to $(6, 0)$
 \rightarrow LEB128 encodes in 2 bytes

COLUMNAR ENCODING (simplified)

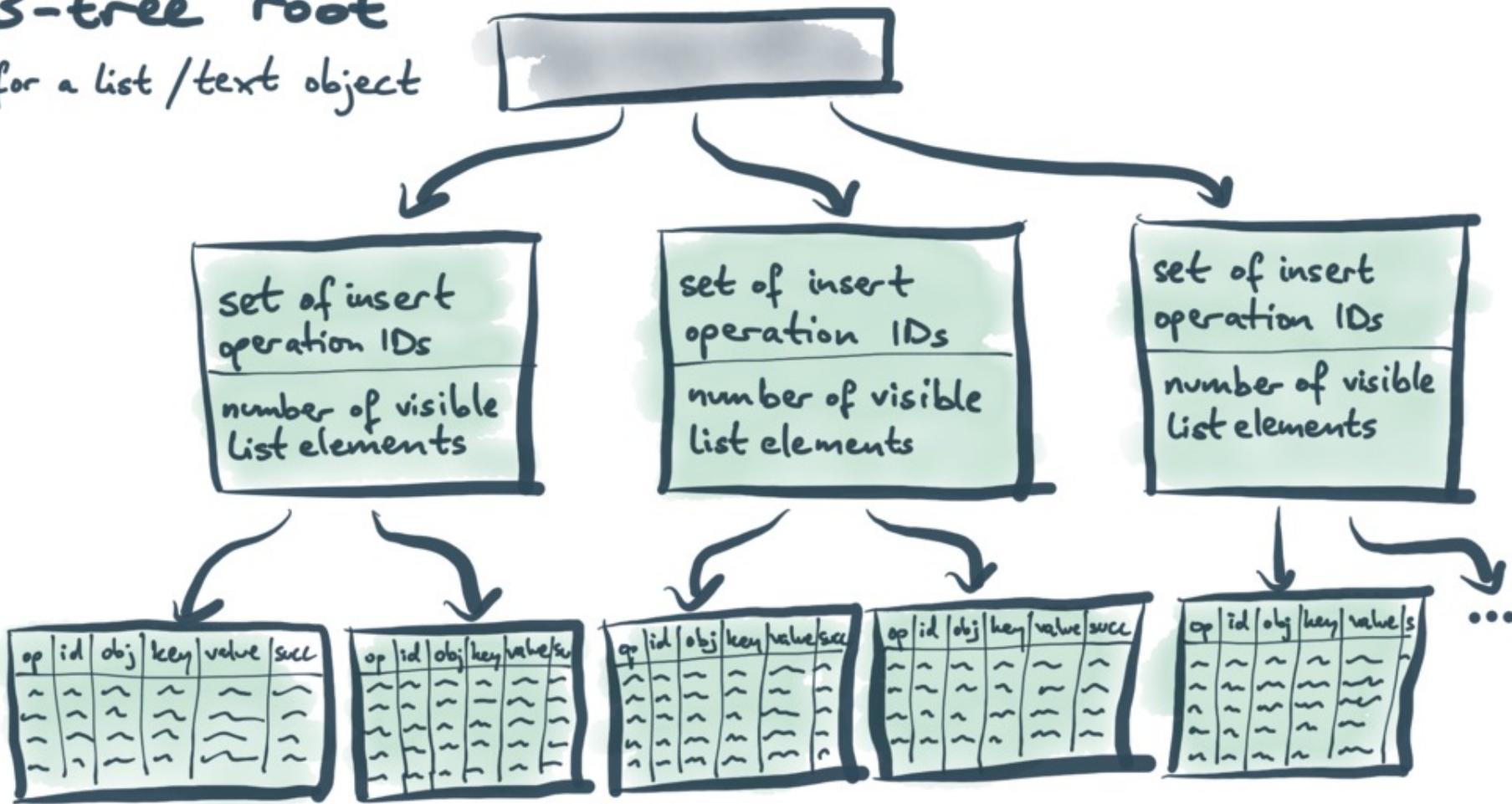
operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	-	-	1	"H"	-	-
2	A	1	A	1	"e"	-	-
3	A	2	A	1	"l"	-	-
4	A	3	A	1	"l"	-	-
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	-	-

just concatenate the UTF-8 ↪

byte sequences → "Hello" (6 bytes)
 (use length column to separate again)

B-tree root

for a list / text object



Local insert: find insertion position by index, counting only visible elements,
to translate index into opID of last insert at that index

Remote insert: find subtree containing position ID, add op to B-tree,
compute index of inserted element based on number of preceding visible elements

Identify document version by version vector

e.g. $V = \{a: 3, b: 4\}$

Visibility rule: operation with $ID = (ctr_{10}, node_{10})$ and
overwrittenBy = $\{(ctr_1, node_1), (ctr_2, node_2), \dots\}$ is visible at document
version V iff $ctr_{10} \leq V[node_{10}]$ and

$\nexists (ctr_i, node_i) \in \text{overwrittenBy}. \quad ctr_i \leq V[node_i].$

Like MVCC in databases with snapshot isolation!

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	-	-	1	"H"	-	-
2	A	1	A	1	"e"	-	-
3	A	2	A	1	"l"	-	-
4	A	3	A	1	"l"	-	-
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	-	-

Identify document version by version vector

e.g. $V = \{a: 3, b: 4\}$

Visibility rule: operation with $ID = (ctr_{10}, node_{10})$ and
overwrittenBy = $\{(ctr_1, node_1), (ctr_2, node_2), \dots\}$ is visible at document
version V iff $ctr_{10} \leq V[node_{10}]$ and

$\nexists (ctr_i, node_i) \in \text{overwrittenBy}. \quad ctr_i \leq V[node_i].$

Like MVCC in databases with snapshot isolation!

For a given key / list element:

- no ops visible \Rightarrow deleted
- one op visible \Rightarrow current value
- multiple ops visible \Rightarrow conflict (concurrent assignment)

Rust
API

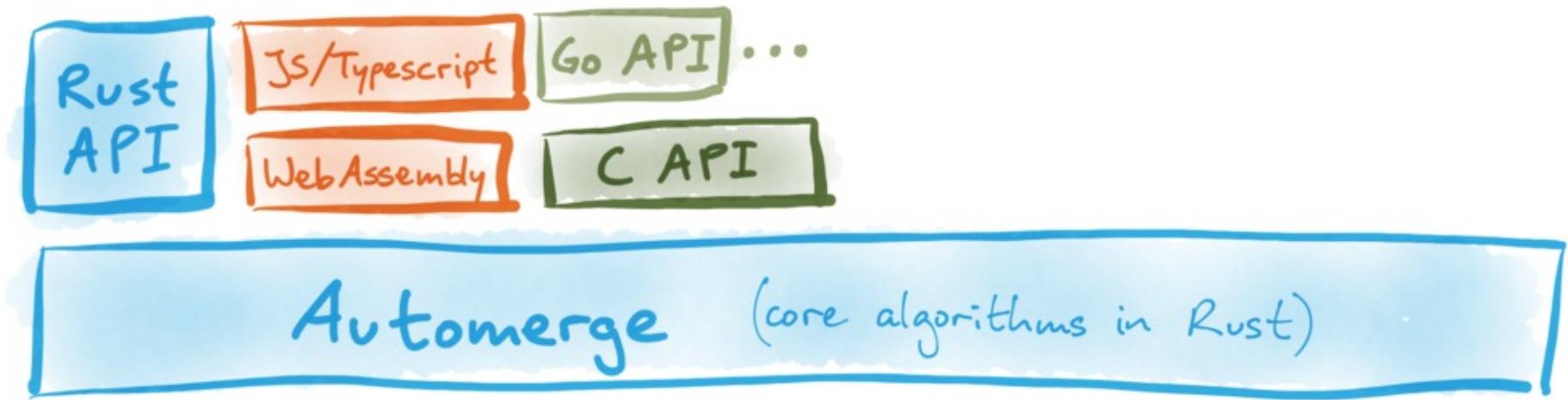
Automerge (core algorithms in Rust)

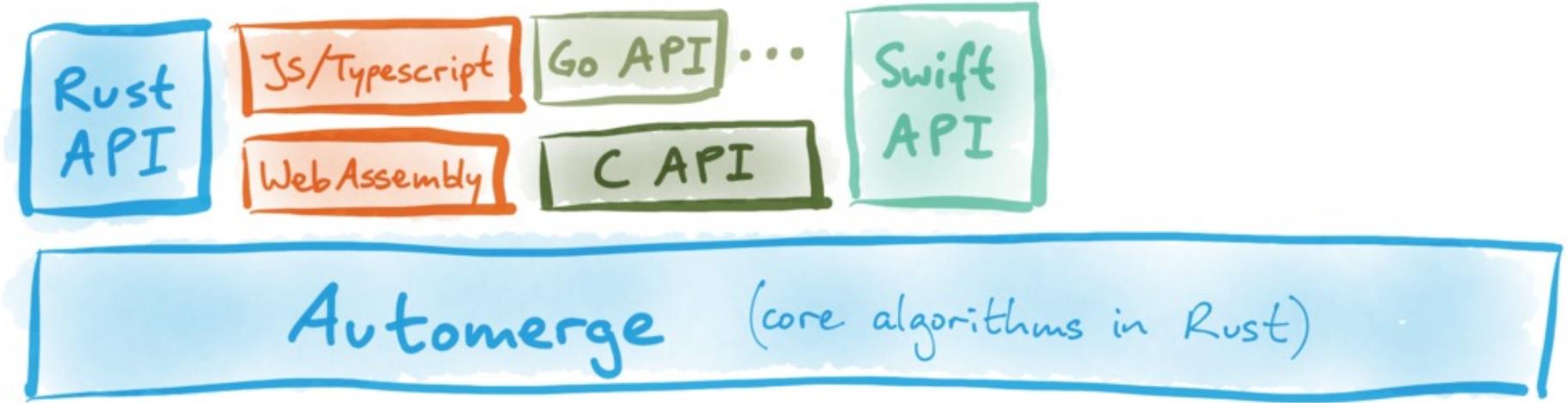
Rust
API

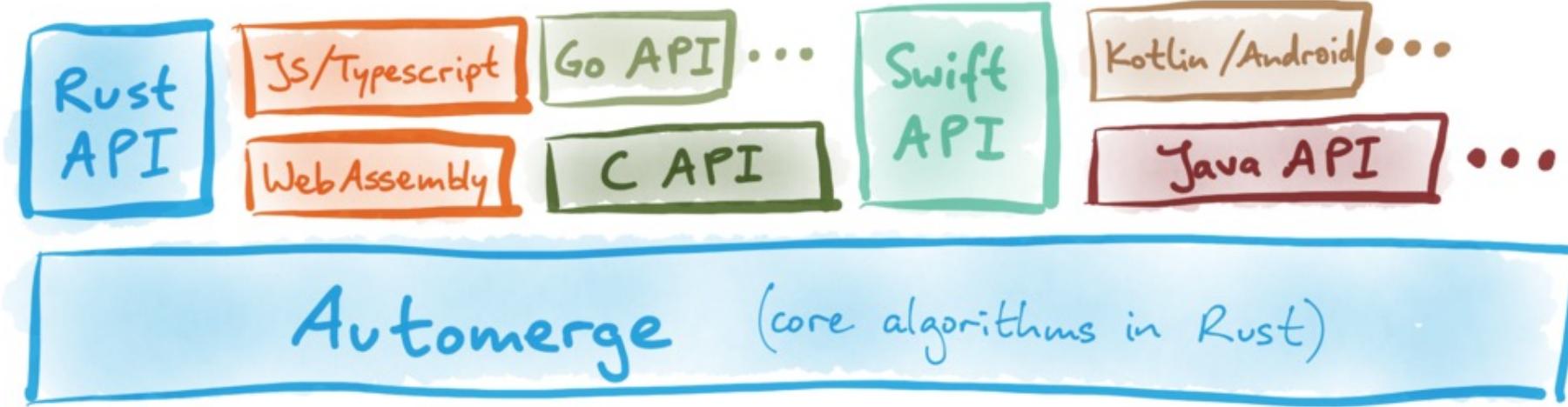
JS/TypeScript

WebAssembly

Automerge (core algorithms in Rust)







Cross-platform apps



Automerger (core algorithms in Rust)

Cross-platform apps



Automerge (core algorithms in Rust)

Automerge-repo (Storage + networking interfaces)

Filesystem IndexedDB •••

storage

Cross-platform apps



Automerge (core algorithms in Rust)

Automerge-repo (Storage + networking interfaces)



A brief history of Automerge

Version 0.1–0.14 (2017–2020)

Research prototype in JavaScript

A brief history of Automerge

Version 0.1–0.14 (2017–2020)

Research prototype in JavaScript

Version 1.0 (2021)

Compressed data format, sync protocol

A brief history of Automerge

Version 0.1–0.14 (2017–2020)

Research prototype in JavaScript

Version 1.0 (2021)

Compressed data format, sync protocol

Version 2.0 (2022)

Moved to Rust, performance, production ready,
commercial support

A brief history of Automerge

Version 0.1–0.14 (2017–2020)

Research prototype in JavaScript

Version 1.0 (2021)

Compressed data format, sync protocol

Version 2.0 (2022)

Moved to Rust, performance, production ready,
commercial support

Version 2.1 (2023)

Rich text, automerge-repo, performance

CLOUD SOFTWARE

Server returns the current state, other copies are just caches

LOCAL-FIRST SOFTWARE

Every device has state
→ Multiple versions explicit
→ Propagating & merging versions

CLOUD SOFTWARE

Server returns the current state, other copies are just caches

Version management + history is an afterthought

LOCAL-FIRST SOFTWARE

Every device has state
→ Multiple versions explicit
→ Propagating & merging versions

OPPORTUNITY:

Build version control into apps at a fundamental level

CLOUD SOFTWARE

Server returns the current state, other copies are just caches

Version management + history is an afterthought

LOCAL-FIRST SOFTWARE

Every device has state
→ Multiple versions explicit

→ Propagating & merging versions

OPPORTUNITY:

Build version control into apps at a fundamental level

Bring Git-like branching/merging (async collaboration) to non-software developers

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text – Peritext
5. Non-interleaving – Fugue

Peritext: A CRDT for Collaborative Rich Text Editing

GEOFFREY LITT, MIT CSAIL, USA

SARAH LIM, UC Berkeley, USA

MARTIN KLEPPMANN, University of Cambridge, United Kingdom

PETER VAN HARDENBERG, Ink & Switch, USA

Conflict-Free Replicated Data Types (CRDTs) support decentralized collaborative editing of shared data, enabling peer-to-peer sharing and flexible branching and merging workflows. While there is extensive work on CRDTs for plain text, much less is known about CRDTs for rich text with formatting. No algorithms have been published, and existing open-source implementations do not always preserve user intent.

In this paper, we describe a model of intent preservation in rich text editing, developed through a series of concurrent editing scenarios. We then describe Peritext, a CRDT algorithm for rich text that satisfies the criteria of our model. The key idea is to store formatting spans alongside the plaintext character sequence, linked to a stable identifier for the first and last character of each span, and then to derive the final formatted text from these spans in a deterministic way that ensures concurrent operations commute.

We have prototyped our algorithm in TypeScript, validated it using randomized property-based testing, and integrated it with an editor UI. We also prove that our algorithm ensures convergence, and demonstrate its causality preservation and intention preservation properties.

CCS Concepts: • Human-centered computing → Asynchronous editors; • Information systems → Version management.

Additional Key Words and Phrases: collaborative editing, asynchronous collaboration, rich text, Conflict-free Replicated Data Types

ACM Reference Format:

Geoffrey Litt, Sarah Lim, Martin Kleppmann, and Peter van Hardenberg. 2022. Peritext: A CRDT for Collaborative Rich Text Editing. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 531 (November 2022), 35 pages.
<https://doi.org/10.1145/3555644>

Rich text as a tree?

A:



The fox jumped.

B:



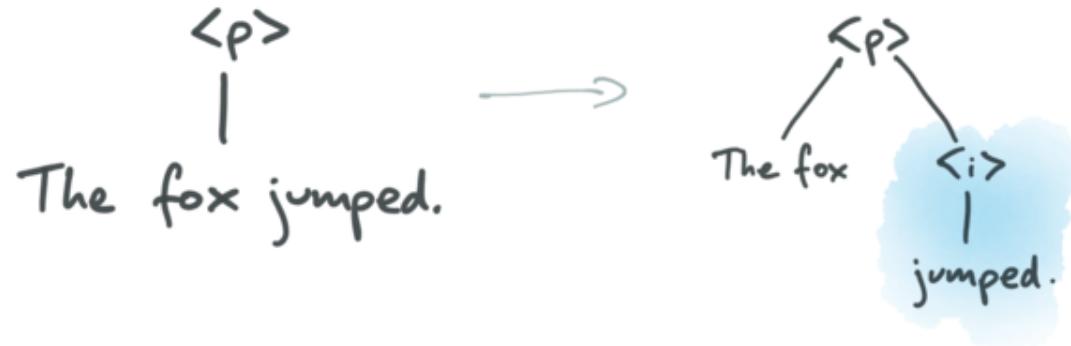
The fox jumped.

Rich text as a tree?

A:

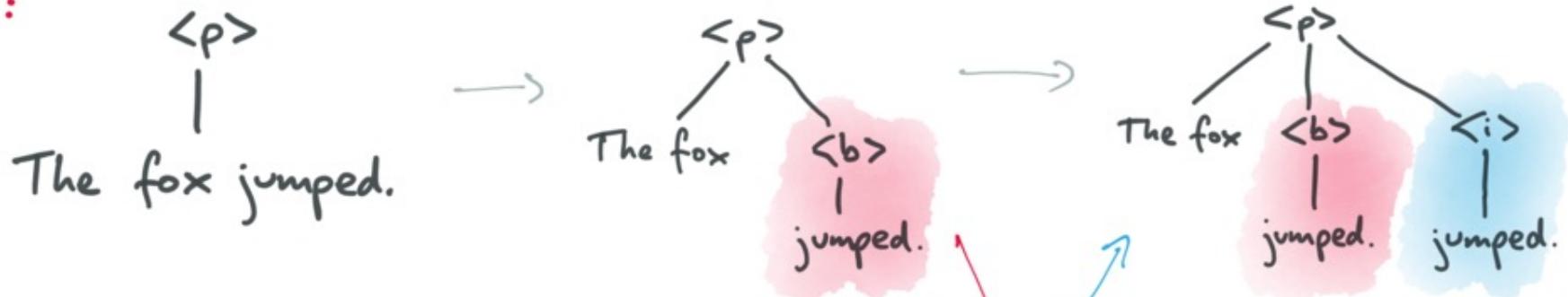


B:



Rich text as a tree?

A:



B:



Rich text as markup?

A:

The fox
jumped
over the dog.

B:

The fox
jumped
over the dog.

Rich text as markup?

A:

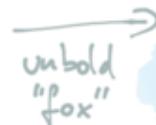
**< b > The fox
jumped
over the dog.**

 unbold everything

The fox jumped
over the dog.

B:

**< b > The fox
jumped
over the dog.**

 unbold "fox"

**< b > The fox
< b > jumped
over the dog.**

Rich text as markup?

A:

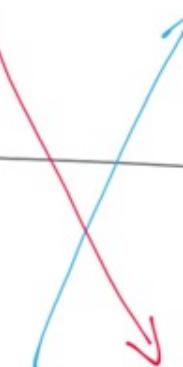
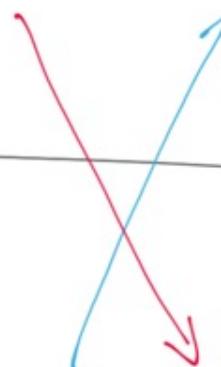
**< b > The fox
jumped
over the dog.**

→
unbold
everything

The fox jumped
over the dog.

→

The fox
**< b > jumped
over the dog.**



B:

**< b > The fox
jumped
over the dog.**

→
unbold
"fox"

**< b > The fox
< b > jumped
over the dog.**

→

The fox
**< b > jumped
over the dog.**

Rich text as per-character properties?

A:

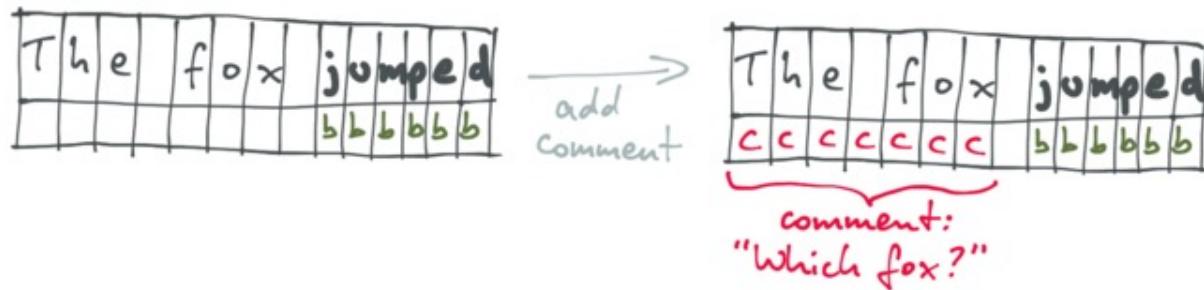
T	h	e		f	o	x	j	u	m	p	e	d
							b	b	b	b	b	b

B:

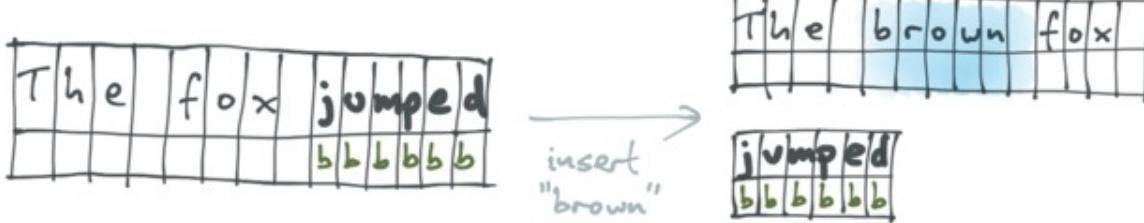
T	h	e		f	o	x	j	u	m	p	e	d
							b	b	b	b	b	b

Rich text as per-character properties?

A:



B:



Rich text as per-character properties?

A:

The fox jumped
b b b b b b

→
add comment

The fox jumped
c c c c c c b b b b b b
comment:
"Which fox?"

The brown fox
ccccccc
Comment:
"Which fox?"
ccc
Comment:
"Which fox?"
jumped
b b b b b b

B:

The fox jumped
b b b b b b

→
insert
"brown"

The brown fox
jumped
b b b b b b

The brown fox
ccccccc
Comment:
"Which fox?"
ccc
Comment:
"Which fox?"
jumped
b b b b b b

Rich text as text with annotation spans

A:

The fox jumped.

t₁: bold

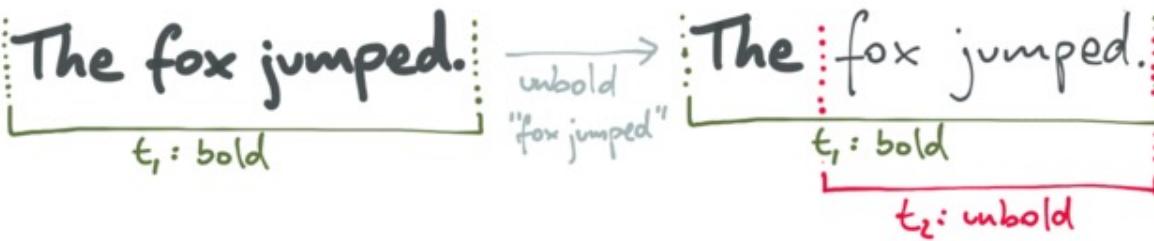
B:

The fox jumped.

t₁: bold

Rich text as text with annotation spans

A:



B:



Rich text as text with annotation spans

A:

The fox jumped.

$t_1: \text{bold}$

→
unbold
"fox jumped"

The fox jumped.

$t_1: \text{bold}$

$t_2: \text{unbold}$

The fox jumped.

$t_1: \text{bold}$

$t_2: \text{unbold}$

$t_3: \text{italic}$

→



B:

The fox jumped.

$t_1: \text{bold}$

→
italicise
"jumped"

The fox jumped.

$t_1: \text{bold}$

$t_3: \text{italic}$

The fox jumped.

$t_1: \text{bold}$

$t_2: \text{unbold}$

$t_3: \text{italic}$

→

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text – Peritext
5. Non-interleaving – Fugue

The Art of the Fugue: Minimizing Interleaving in Collaborative Text Editing

Matthew Weidner and Martin Kleppmann

Abstract—Most existing algorithms for replicated lists, which are widely used in collaborative text editors, suffer from a problem: when two users concurrently insert text at the same position in the document, the merged outcome may interleave the inserted text passages, resulting in corrupted and potentially unreadable text. The problem has gone unnoticed for decades, and it affects both CRDTs and Operational Transformation. This paper presents Fugue, an algorithm that guarantees maximal non-interleaving, our new correctness property for replicated lists. We also implement Fugue and demonstrate that it offers performance comparable to state-of-the-art CRDT libraries for text editing.

Index Terms—Distributed data structures, replica consistency, collaborative text editing, Conflict-free Replicated Data Types (CRDTs), operational transformation.

1 INTRODUCTION

COLLABORATIVE text editors such as Google Docs allow several users to concurrently modify a document, while ensuring that all users' copies of the document converge to the same state. This type of software is implemented as a replicated list of characters; a replicated list object is a distributed data structure whose semantics can be formally specified [1].

Even though algorithms for replicated lists and collaborative text editing have been studied for over three decades [2], [3], [4], a formal specification of the required behavior of a replicated list only appeared as recently as 2016 [1]. We argue that this specification is incomplete. In Section 2 we show that there is an additional correctness property that is important in practice, but which has been overlooked by almost all prior research on this topic: *non-interleaving*. Informally stated, this property requires that when sections of text are composed independently from

proposed algorithm in that paper does not converge (Section 2.4).

- We extend the formal specification of replicated lists by Attiya et al. [4] with a new property, which we call *maximal non-interleaving* (Section 5.1). This definition is subtle: we show that an alternative, simpler definition is also impossible to satisfy (Section 5.1).
- We introduce *Fugue*, a CRDT algorithm for replicated lists that (to our knowledge) is the first algorithm proven to satisfy maximal non-interleaving (Section 3).
- We provide an optimized open source implementation of Fugue, and show that it achieves memory, network, and CPU performance comparable to the state-of-the-art Yjs library on a realistic text-editing trace (Section 4).

m	i	l	k	\n
0.21	0.32	0.46	0.66	0.91

User A

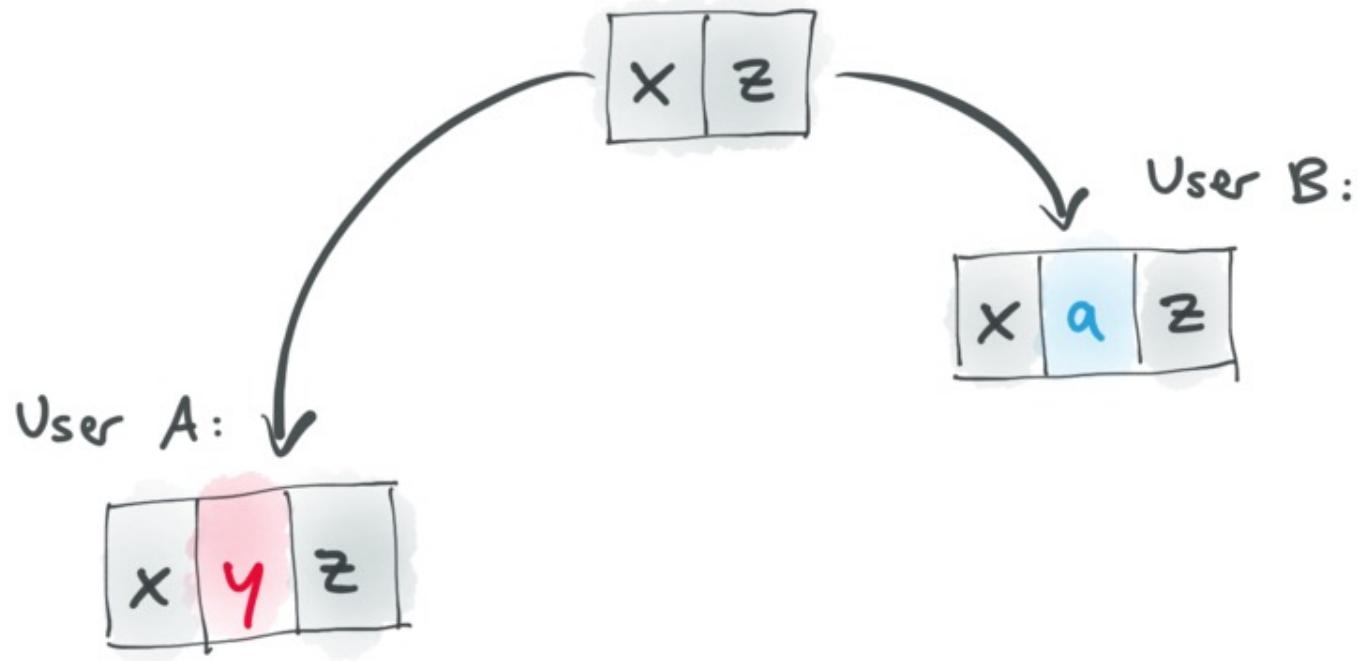
m	i	l	k	\n	e	g	g	s	\n
0.21	0.32	0.46	0.66	0.70	0.74	0.79	0.83	0.86	0.91

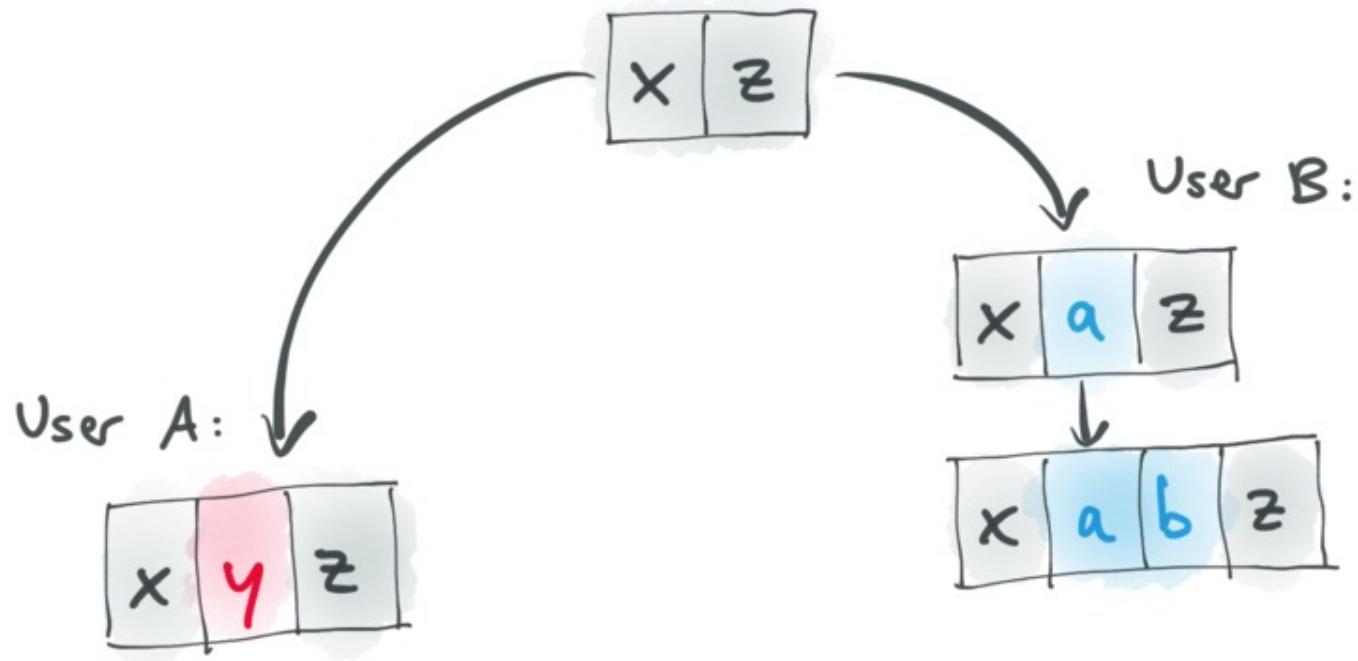
User B

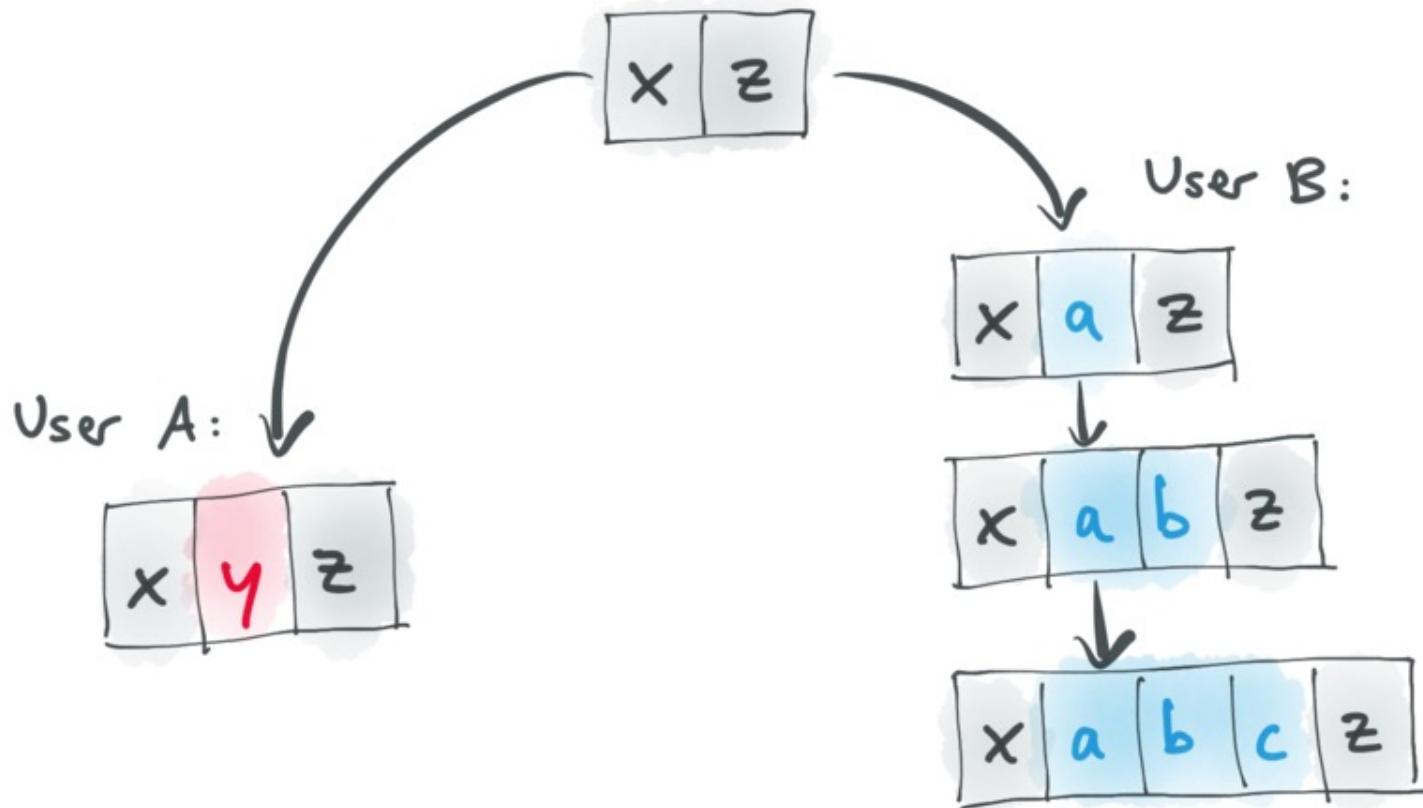
m	i	l	k	\n	b	r	e	a	d	\n
0.21	0.32	0.46	0.66	0.72	0.75	0.77	0.80	0.84	0.88	0.91

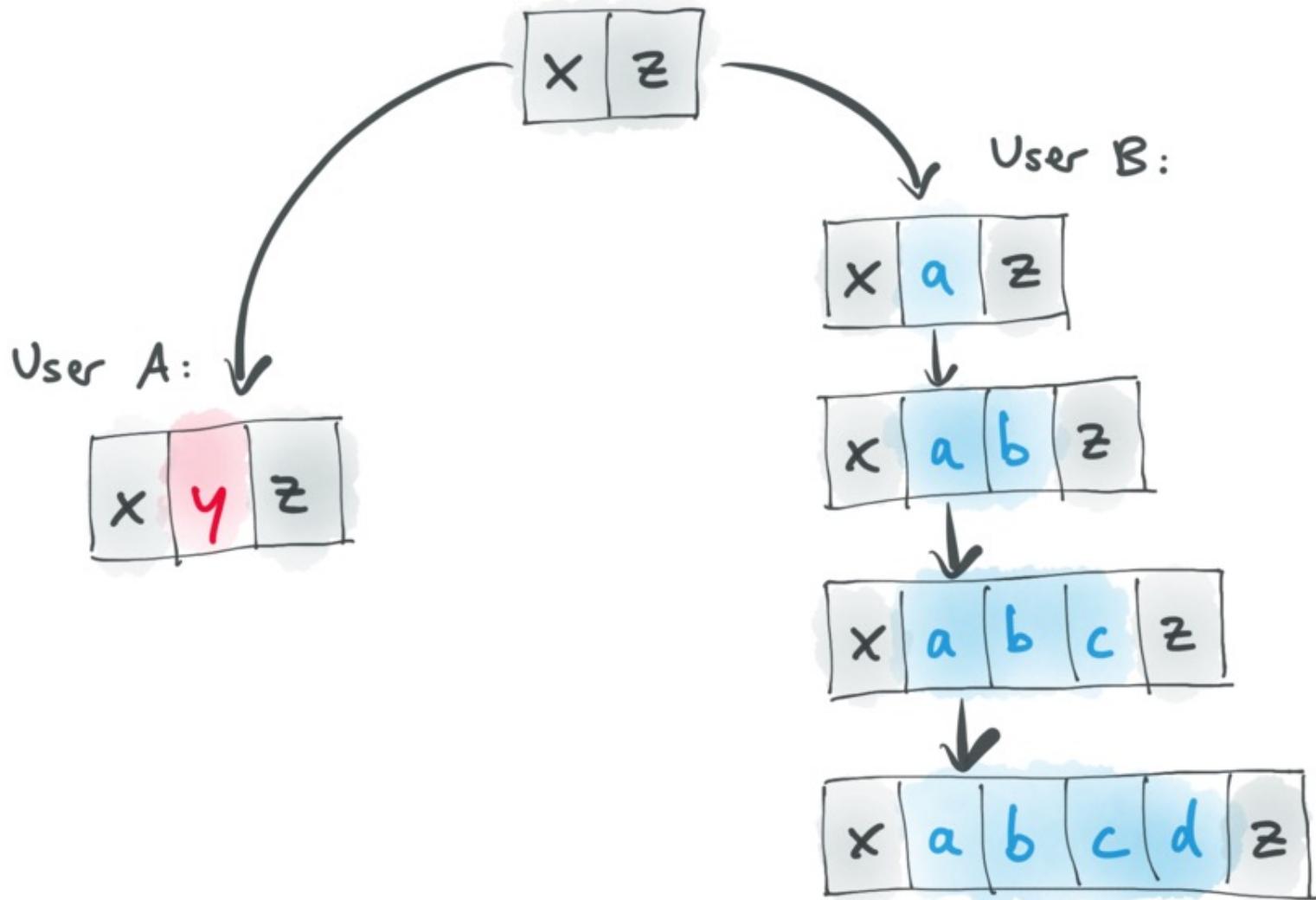
merged:

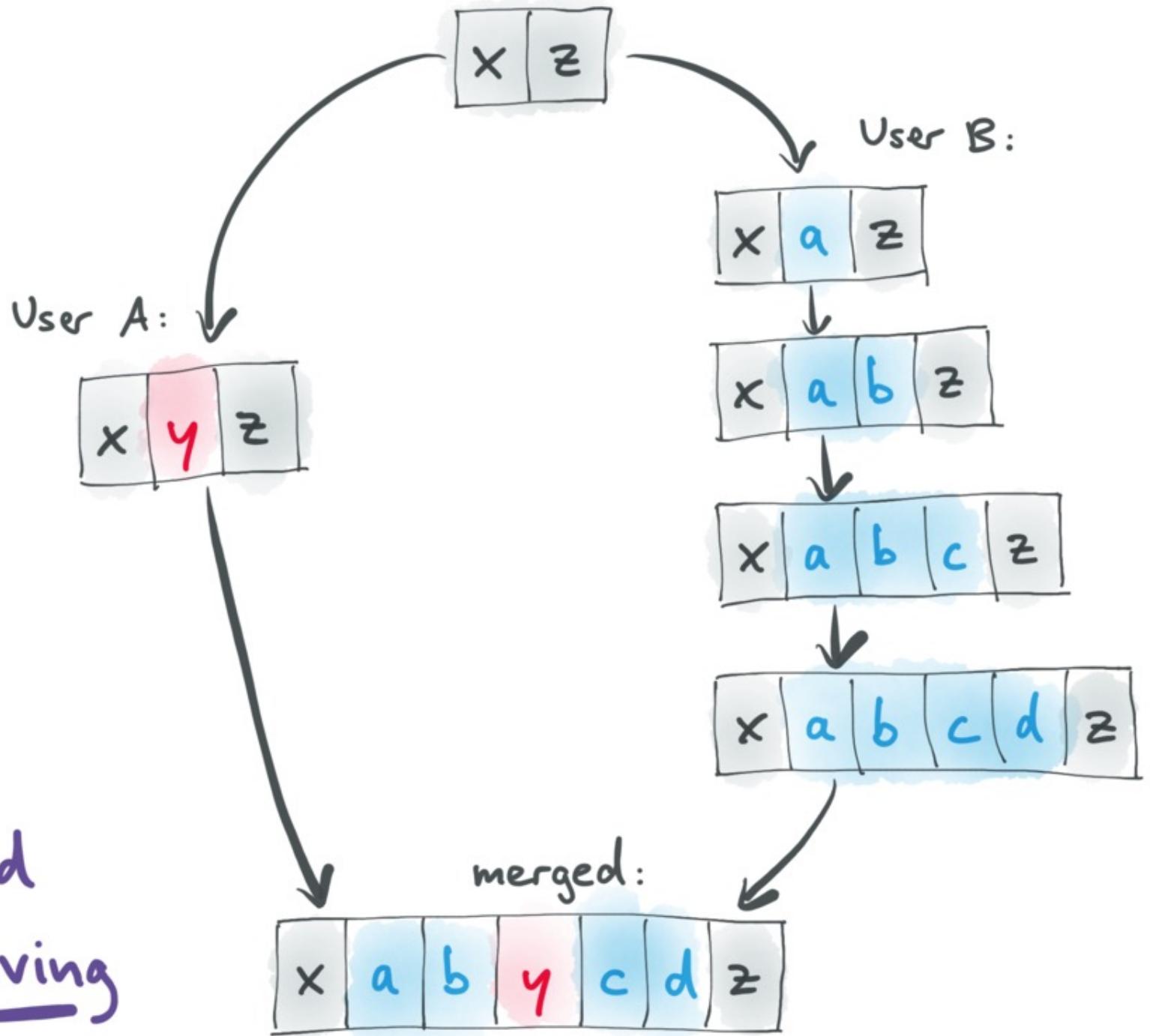
m	i	l	k	\n	\n	e	b	r	g	g	a	s	\n		
0.21	0.32	0.46	0.66	0.70	0.72	0.74	0.75	0.77	0.79	0.80	0.83	0.84	0.86	0.88	0.91

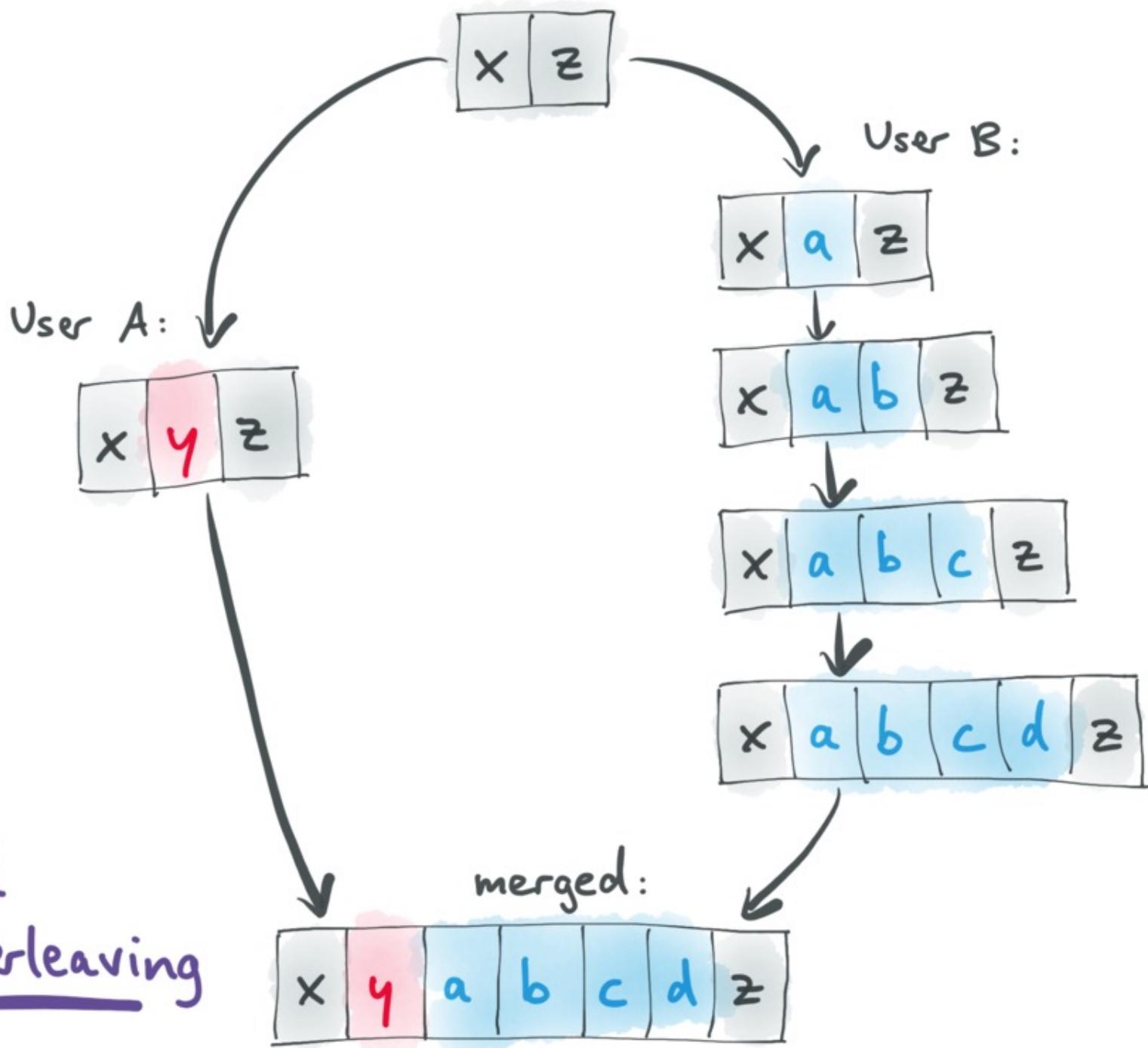


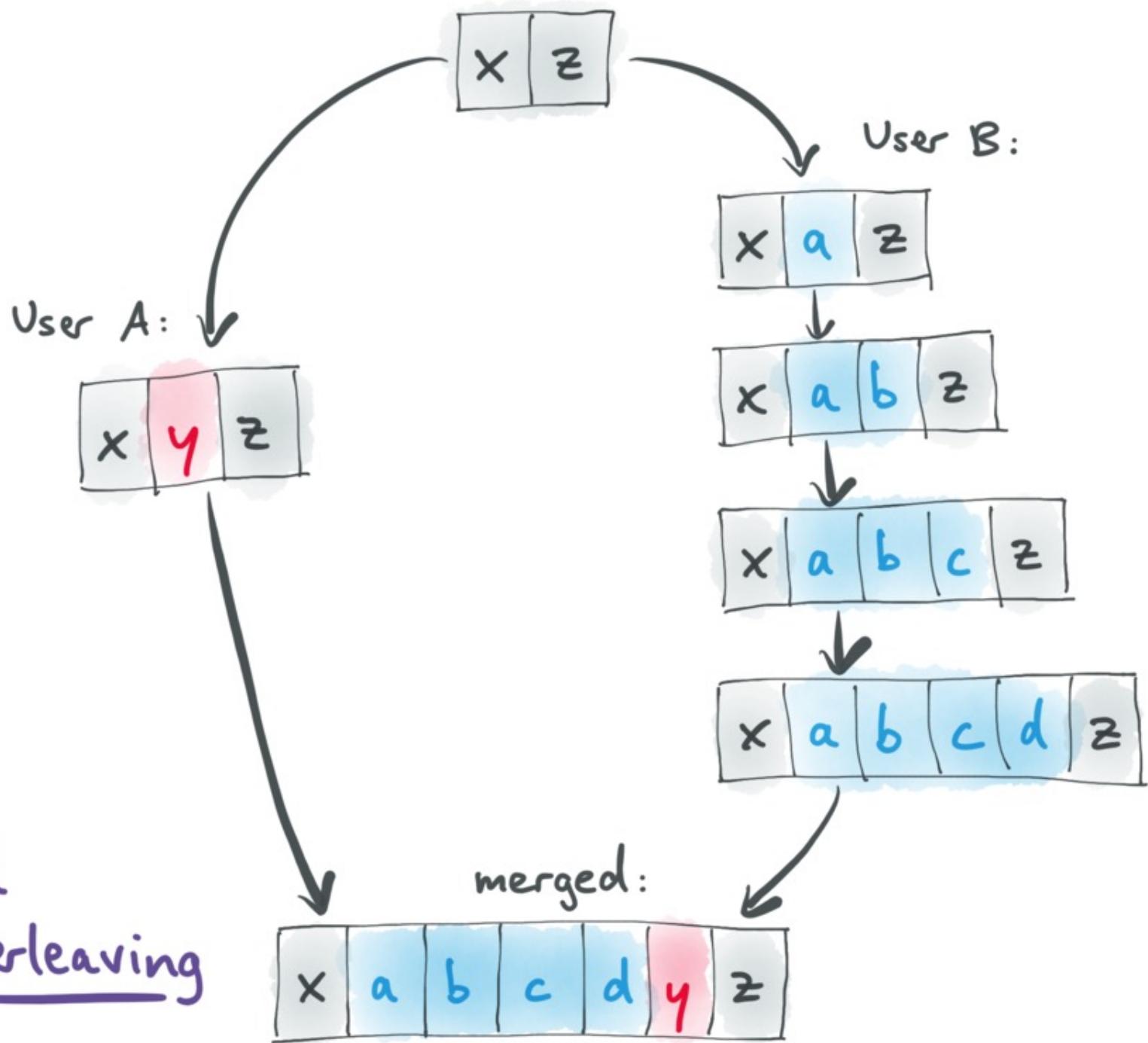




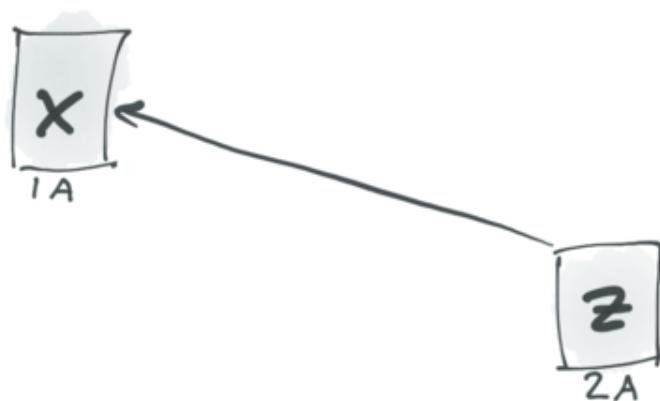




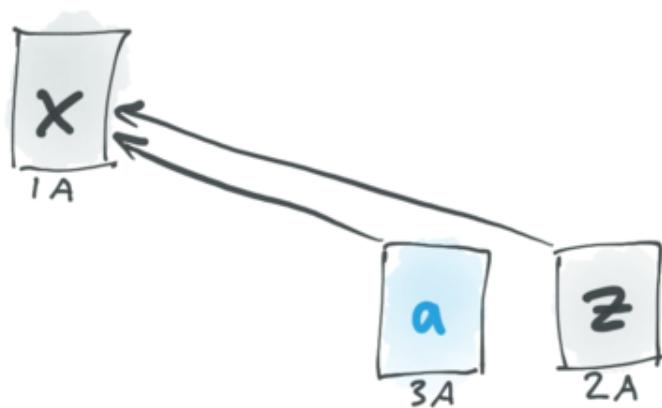




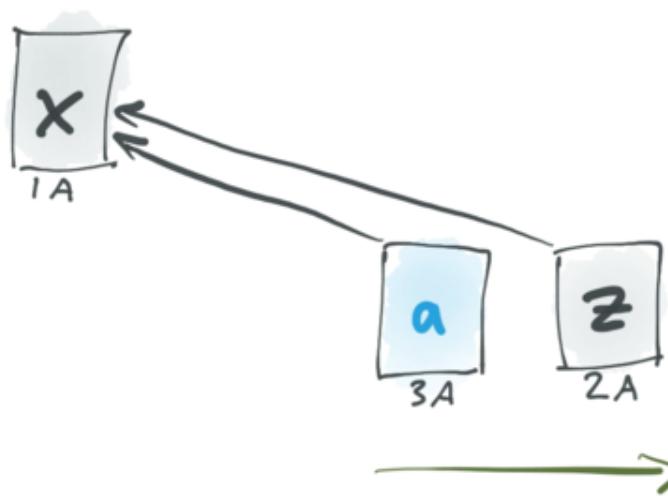
RGA is forward non-interleaving



RGA is forward non-interleaving

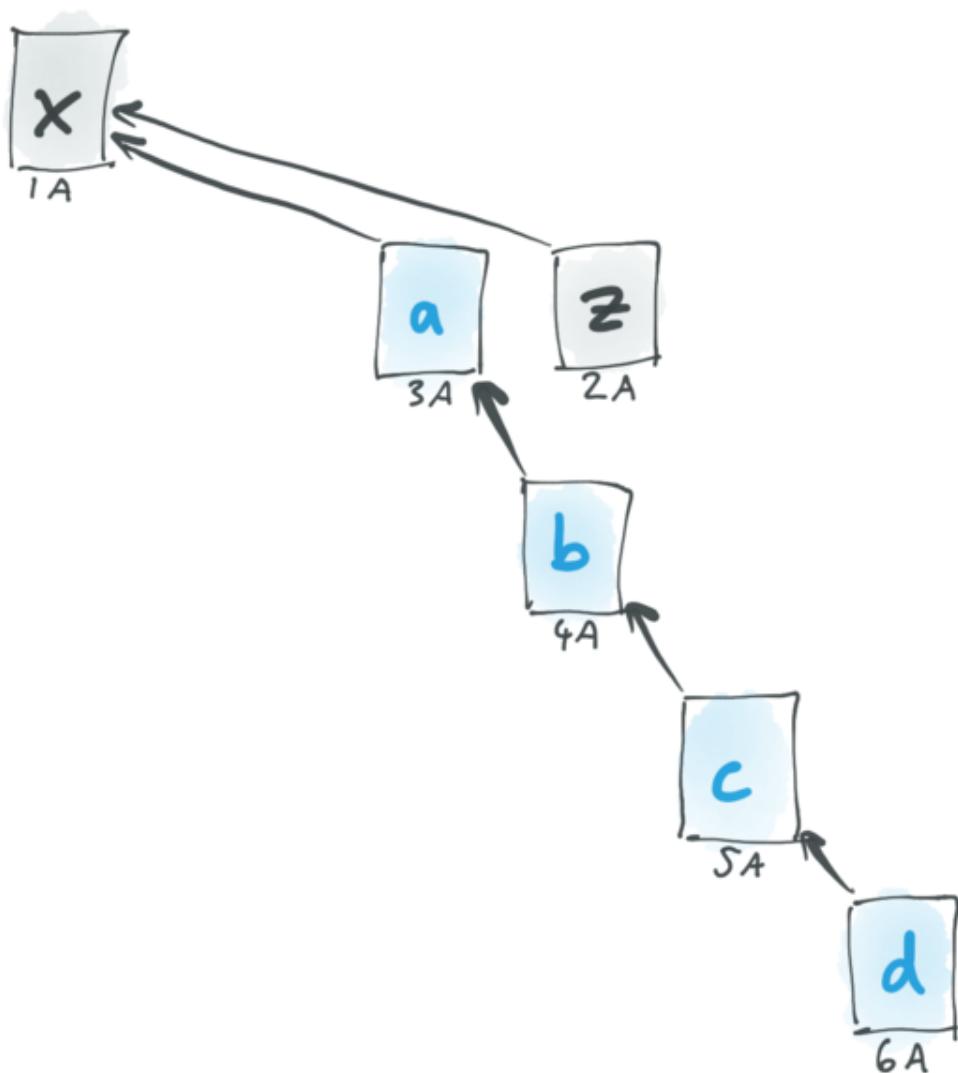


RGA is forward non-interleaving

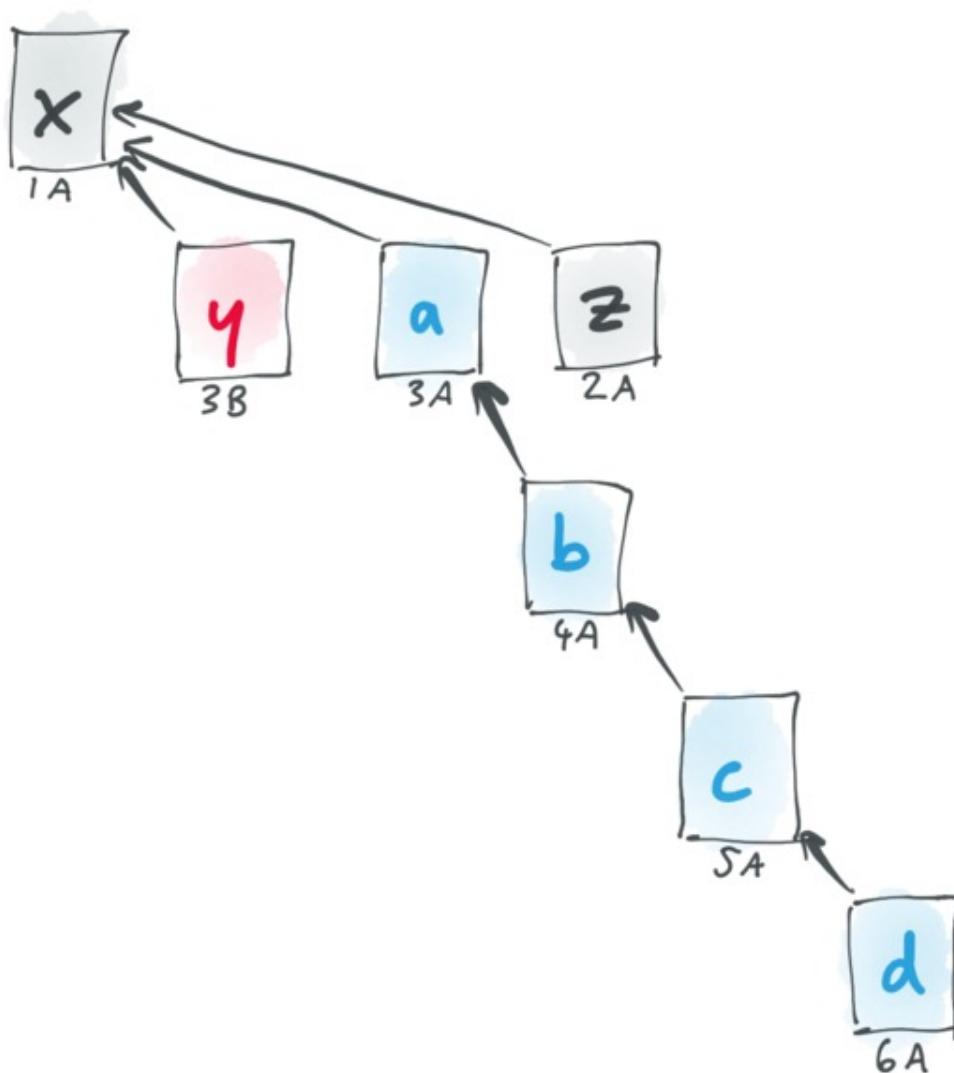


visit siblings in descending timestamp order

RGA is forward non-interleaving



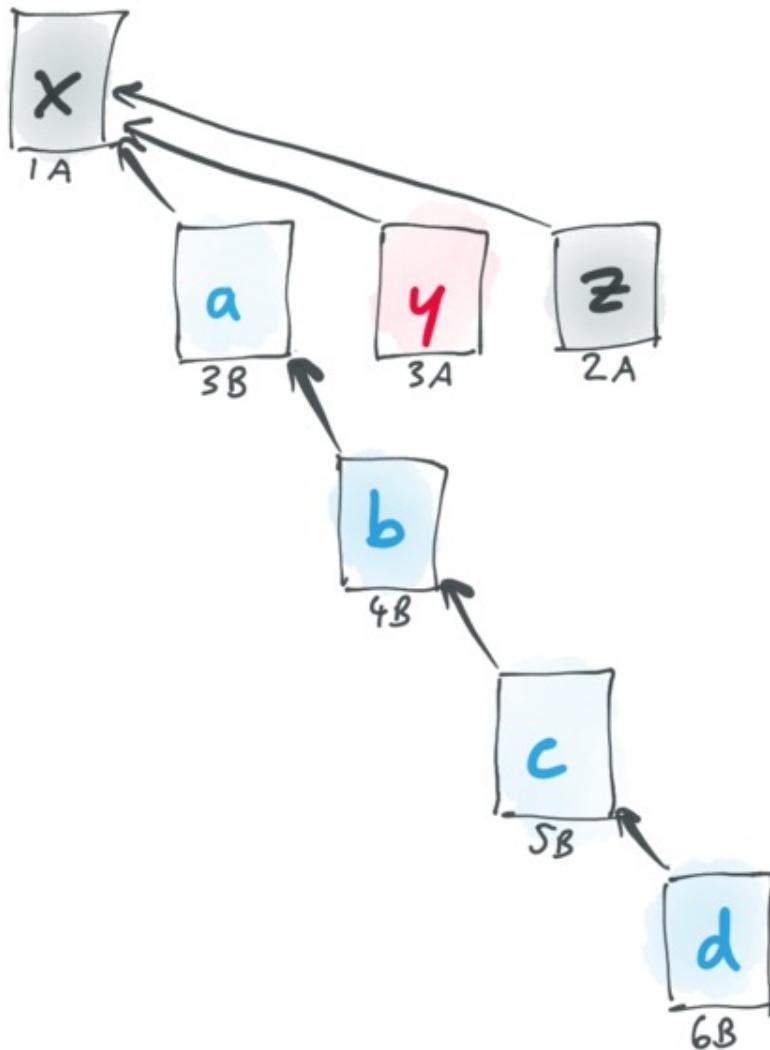
RGA is forward non-interleaving



Depth-first traversal order:

x	y	a	b	c	d	z
1A	3B	3A	4A	5A	6A	2A

RGA is forward non-interleaving



Depth-first traversal order:

x	a	b	c	d	y	z
1A	3B	4B	5B	6B	3A	2A

User A:

x	y	z
---	---	---

x	z
---	---

User B:

x	d	z
---	---	---

User A:

x	y	z
---	---	---

x	z
---	---

User B:

x	d	z
---	---	---

x	c	d	z
---	---	---	---

User A:

x	y	z
---	---	---

x	z
---	---

User B:

x	d	z
---	---	---

x	c	d	z
---	---	---	---

x	b	c	d	z
---	---	---	---	---

User A:

x	y	z
---	---	---

x	z
---	---

User B:

x	d	z
---	---	---

x	c	d	z
---	---	---	---

x	b	c	d	z
---	---	---	---	---

x	a	b	c	d	z
---	---	---	---	---	---

User A:

x	y	z
---	---	---

x	z
---	---

User B:

x	d	z
---	---	---

x	c	d	z
---	---	---	---

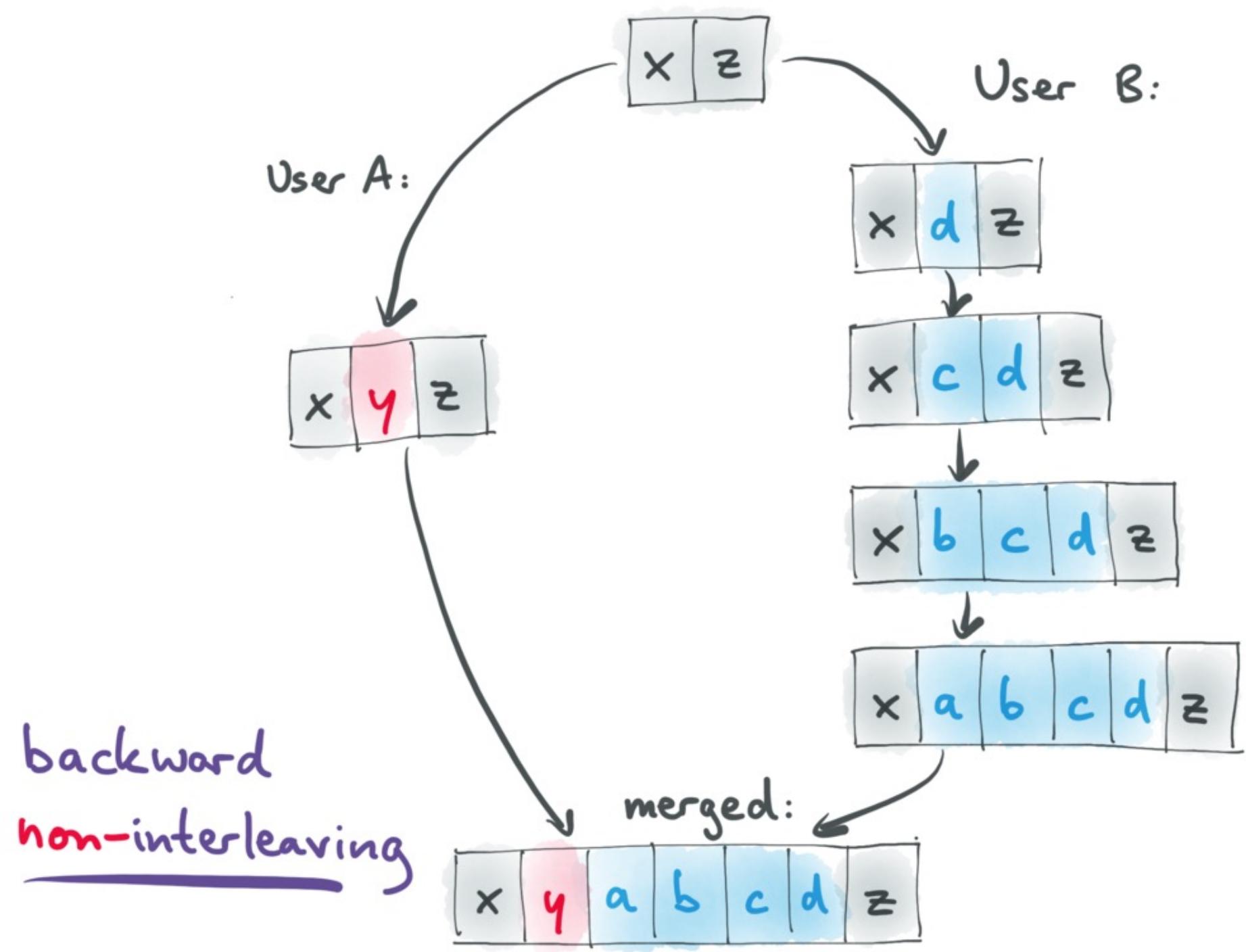
x	b	c	d	z
---	---	---	---	---

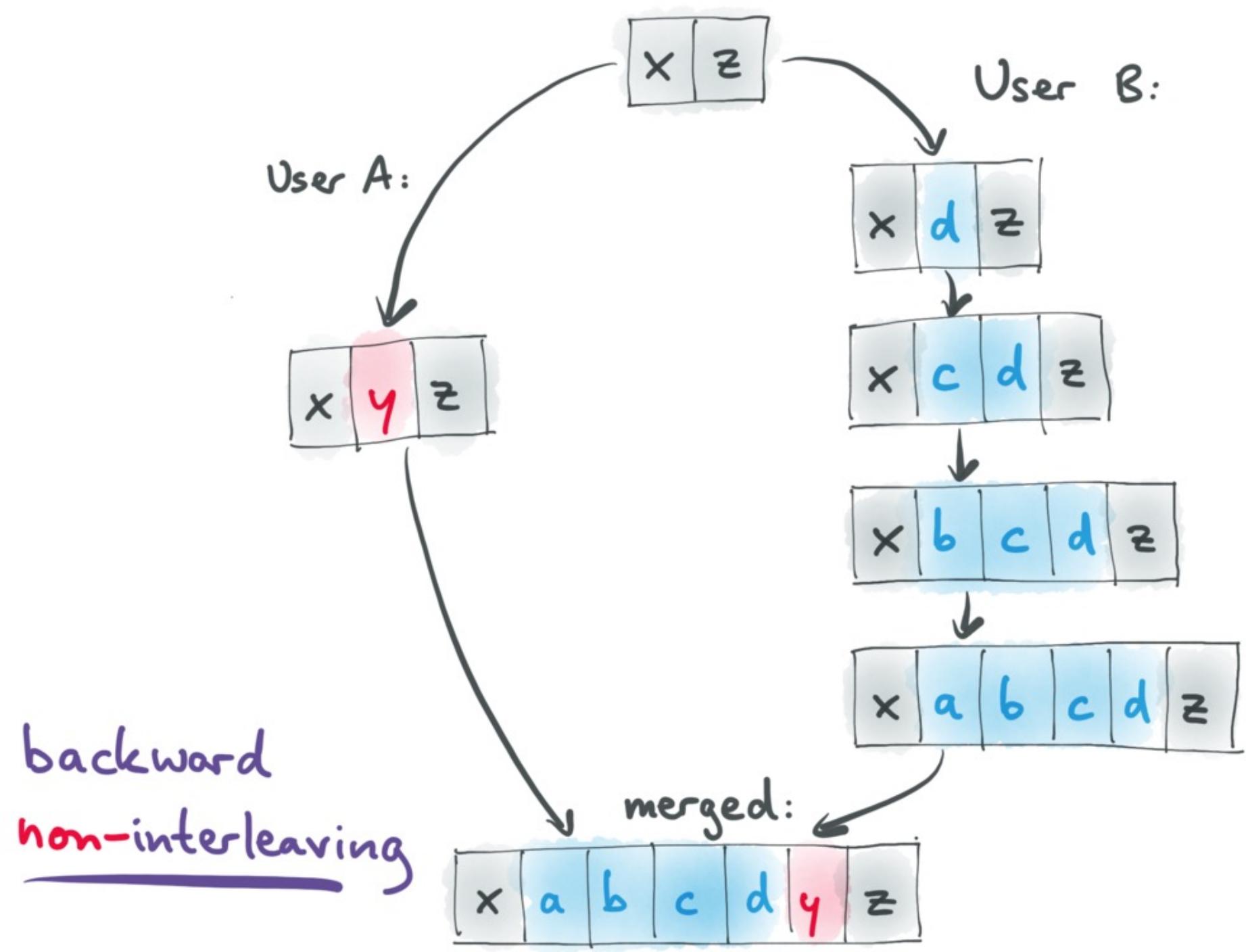
x	a	b	c	d	z
---	---	---	---	---	---

backward interleaving

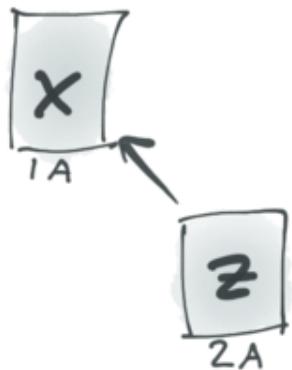
merged:

x	a	b	y	c	d	z
---	---	---	---	---	---	---

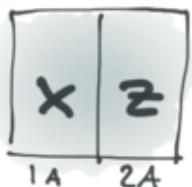




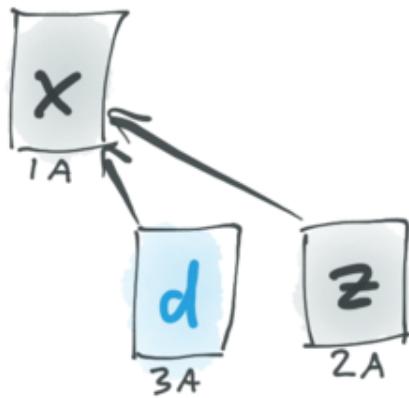
Backward interleaving in RGA



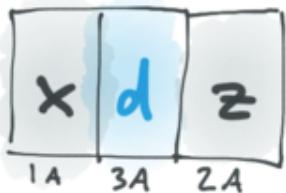
Depth-first traversal order:



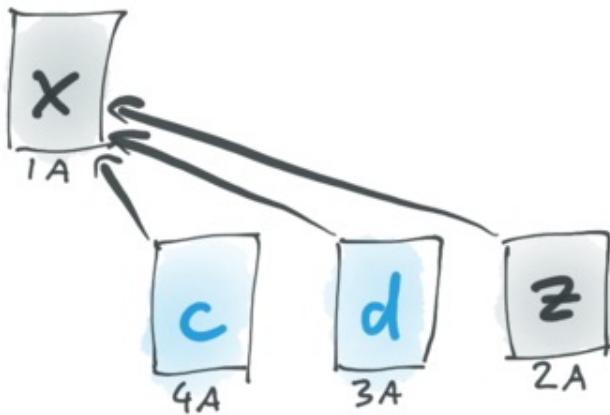
Backward interleaving in RGA



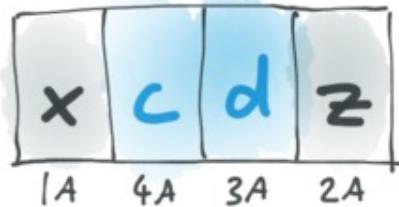
Depth-first traversal order:



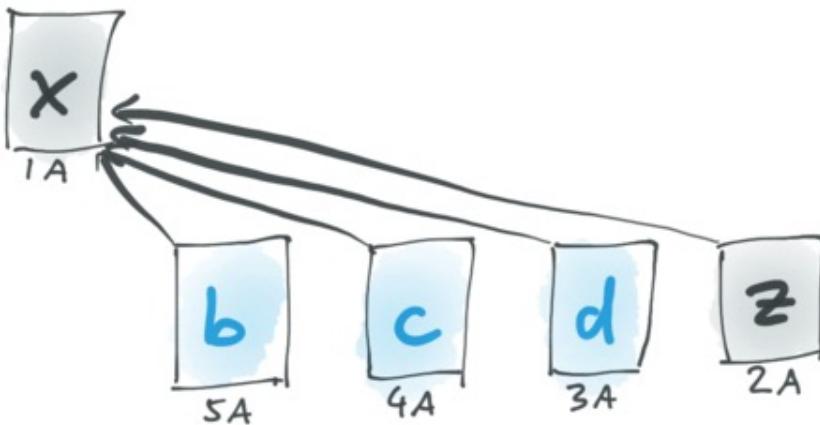
Backward interleaving in RGA



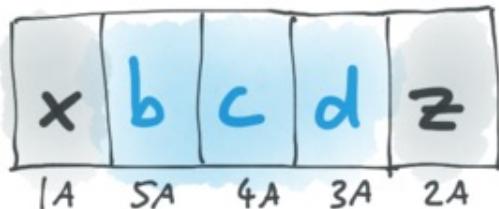
Depth-first traversal order:



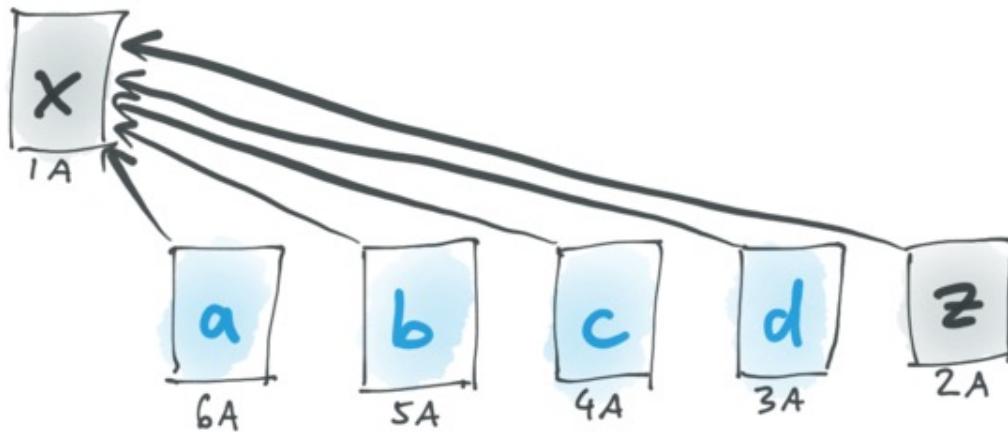
Backward interleaving in RGA



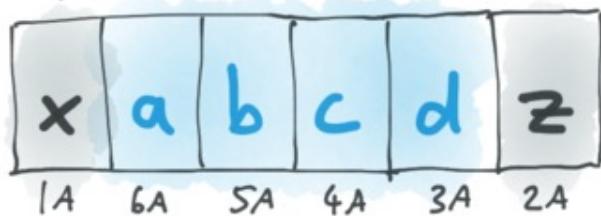
Depth-first traversal order:



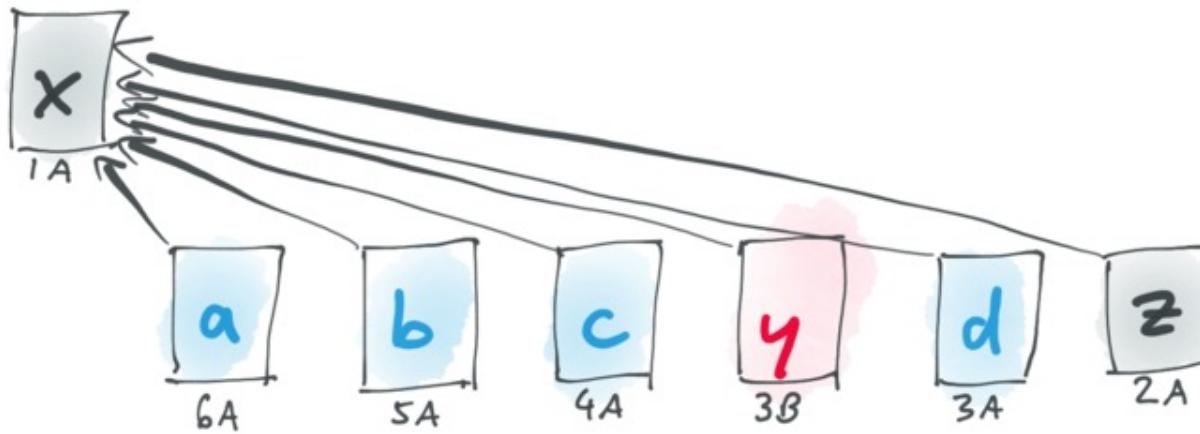
Backward interleaving in RGA



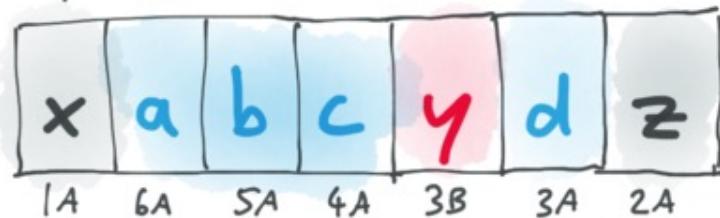
Depth-first traversal order:



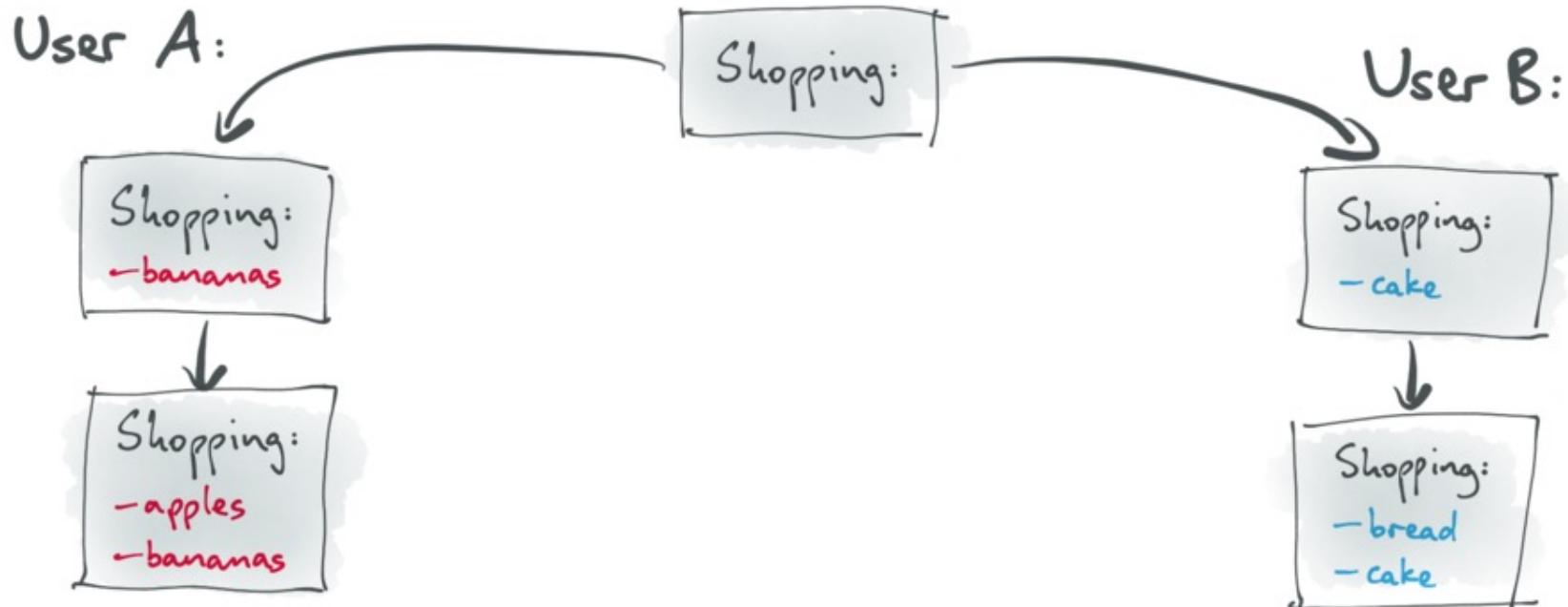
Backward interleaving in RGA

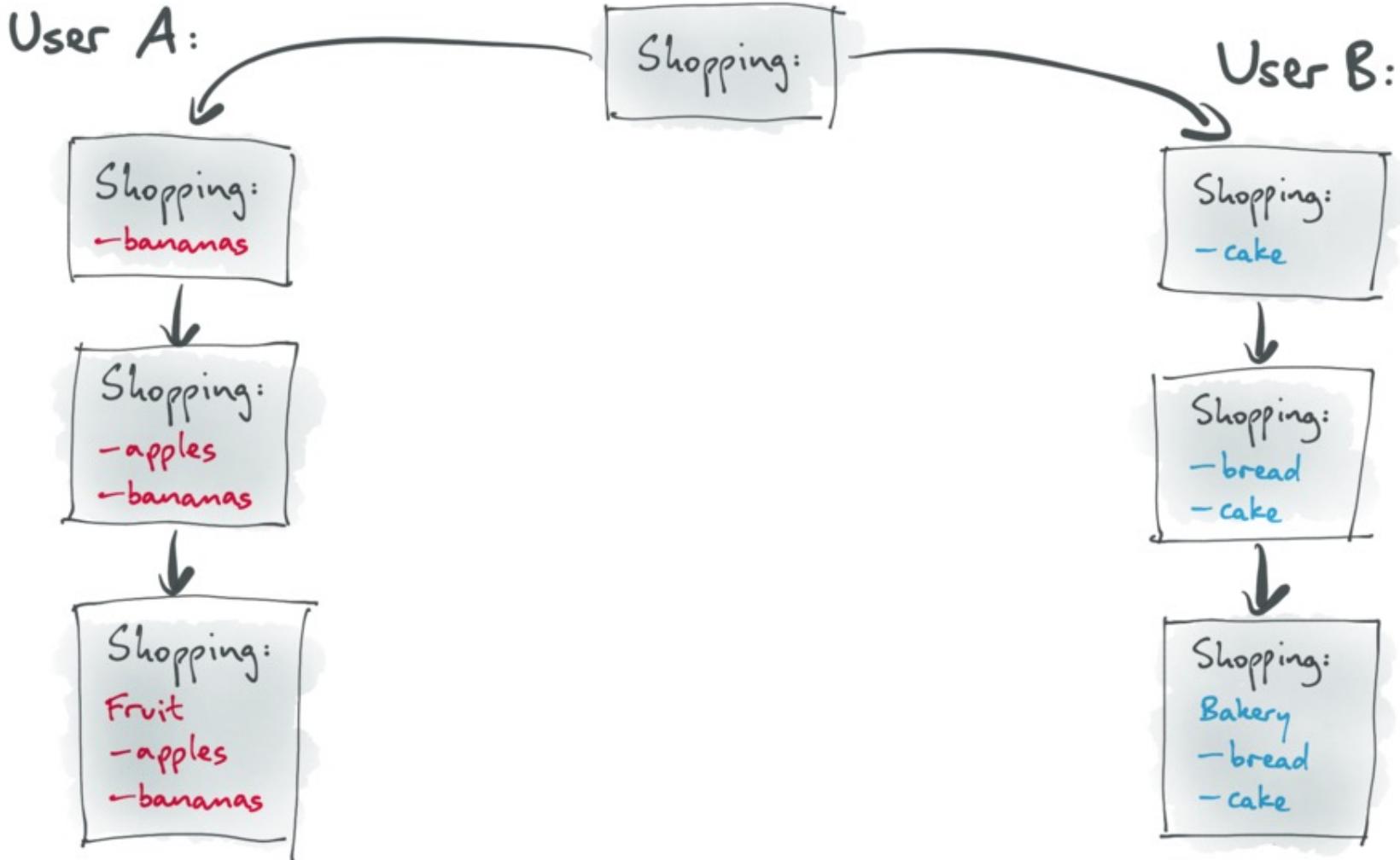


Depth-first traversal order:









User A: User B:

Shopping:
-bananas

Shopping:
-apples
-bananas

Shopping:
Fruit
-apples
-bananas

Shopping:

Shopping:
-cake

Shopping:
-bread
-cake

Shopping:
Bakery
-bread
-cake

backward interleaving

merged:

Shopping:
Fruit
Bakery
-apples
-bread
-bananas
-cake

User A:

Shopping:
-bananas

backward
non-interleaving

User B:

Shopping:
-cake

backward
non-interleaving

Shopping:
-apples
-bananas

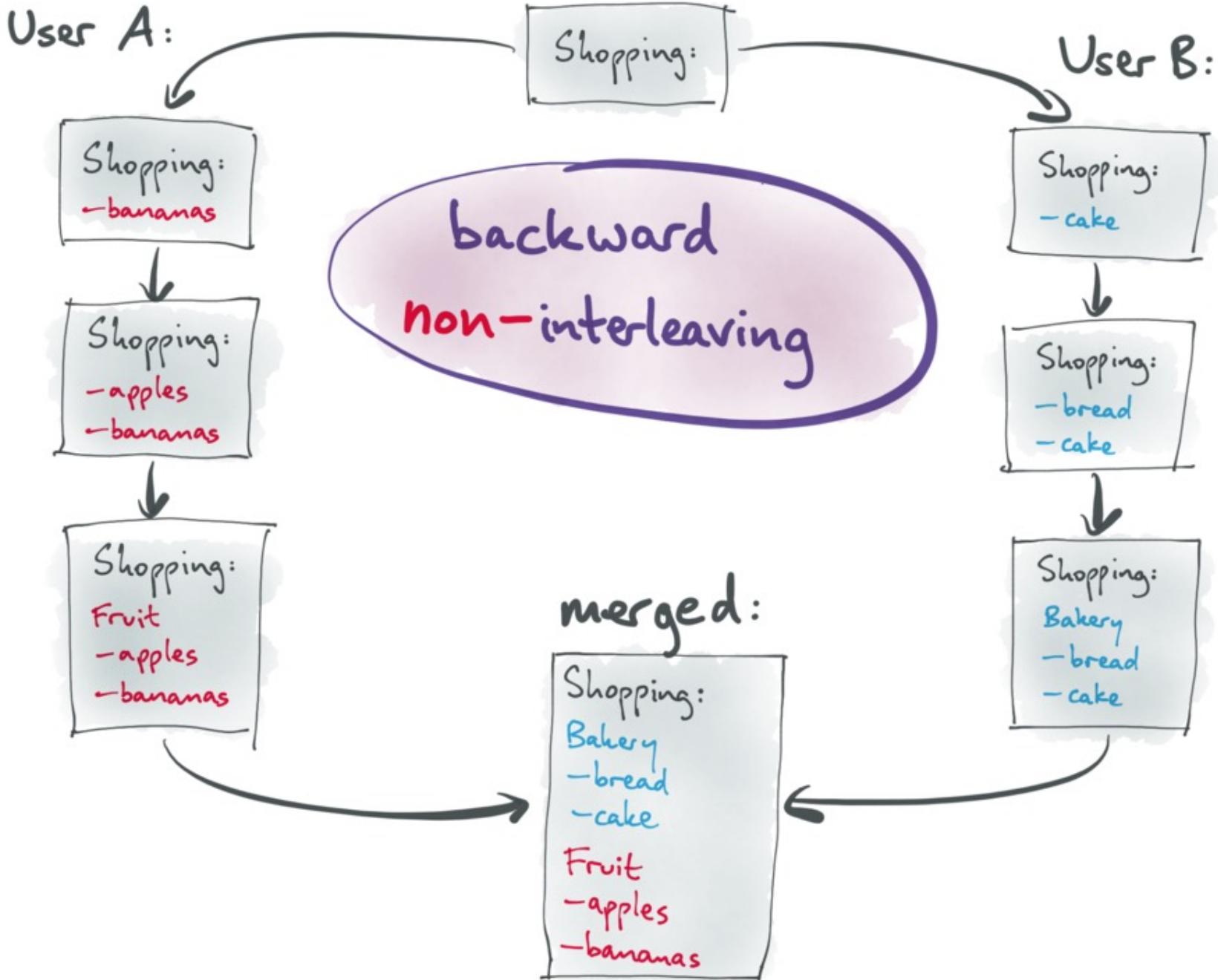
Shopping:
-bread
-cake

Shopping:
Fruit
-apples
-bananas

Shopping:
Bakery
-bread
-cake

merged:

Shopping:
Fruit
-apples
-bananas
Bakery
-bread
-cake



■ **Table 1** Various algorithms' susceptibility to interleaving anomalies. Key: ● = interleaving can occur; ○ = we have not been able to find examples of interleaving; ○✓ = proven not to interleave; ⚡ = algorithm may incorrectly reorder characters. Examples of anomalies appear in Appendix A.

Family	Algorithm	forward interleaving (one replica)	forward interleaving (multi-replica)	backward interleaving (one replica)	backward interleaving (multi-replica)
OT	adOPTed [25]	●	●	○	●
	Jupiter [18]	●	●	○	○
	GOT [31]	●	●	⚡	⚡
	SOCT2 [29]	●	●	●	●
	TTF [20]	●	●	○	●
CRDT	WOOT [22]	●	●	○	○
	Logoot [35]	●	●	●	●
	LSEQ [17]	●	●	●	●
	Treedoc [24]	●	●	●	●
	RGA [26]	○✓	○✓	●	●
	Yjs [10]	○✓	○✓	○	●
	Fugue (this work)	○✓	○✓	○✓	○✓

RGA is non-interleaving for forward insertions,
but interleaves backward insertions.

Could swap forward/backward by inserting before ID, not after.

RGA is non-interleaving for forward insertions,
but interleaves backward insertions.

Could swap forward/backward by inserting before ID, not after.

Can we make an algorithm that is non-interleaving
for both forward and backward insertions?

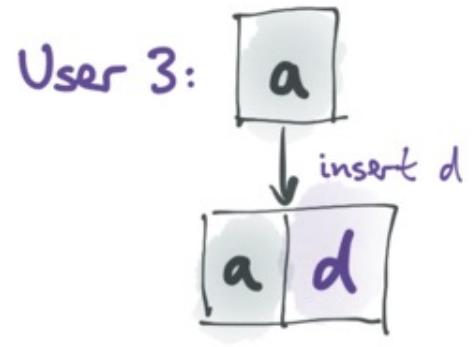
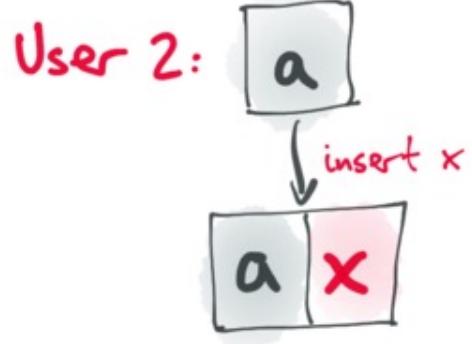
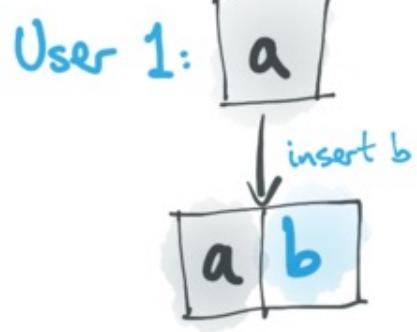
RGA is non-interleaving for forward insertions,
but interleaves backward insertions.

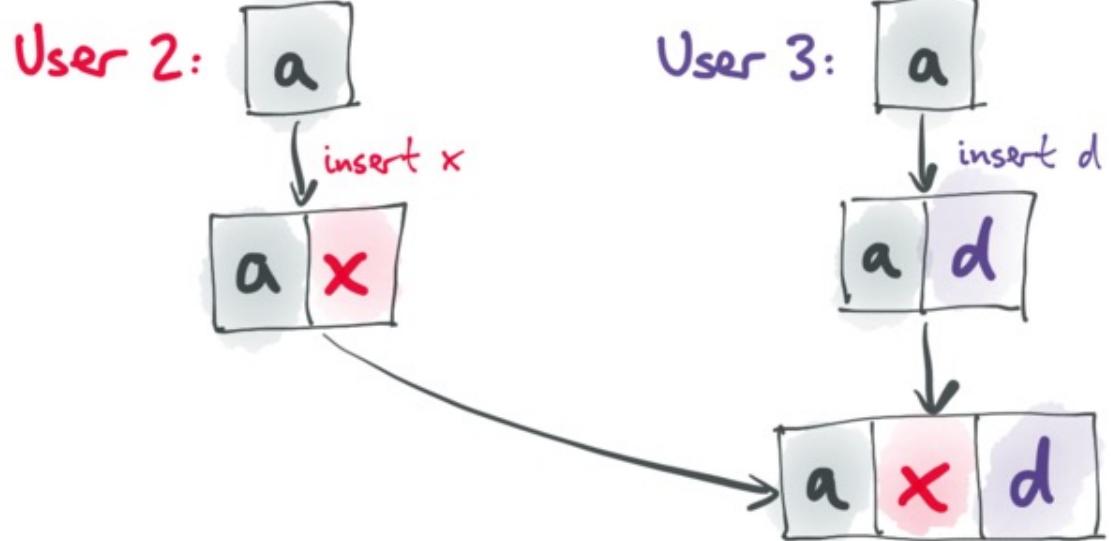
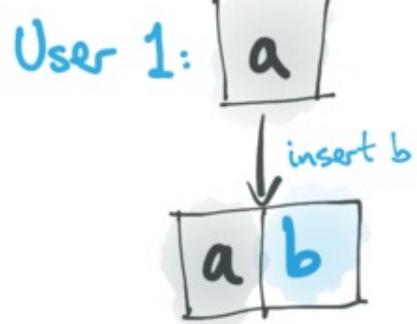
Could swap forward/backward by inserting before ID, not after.

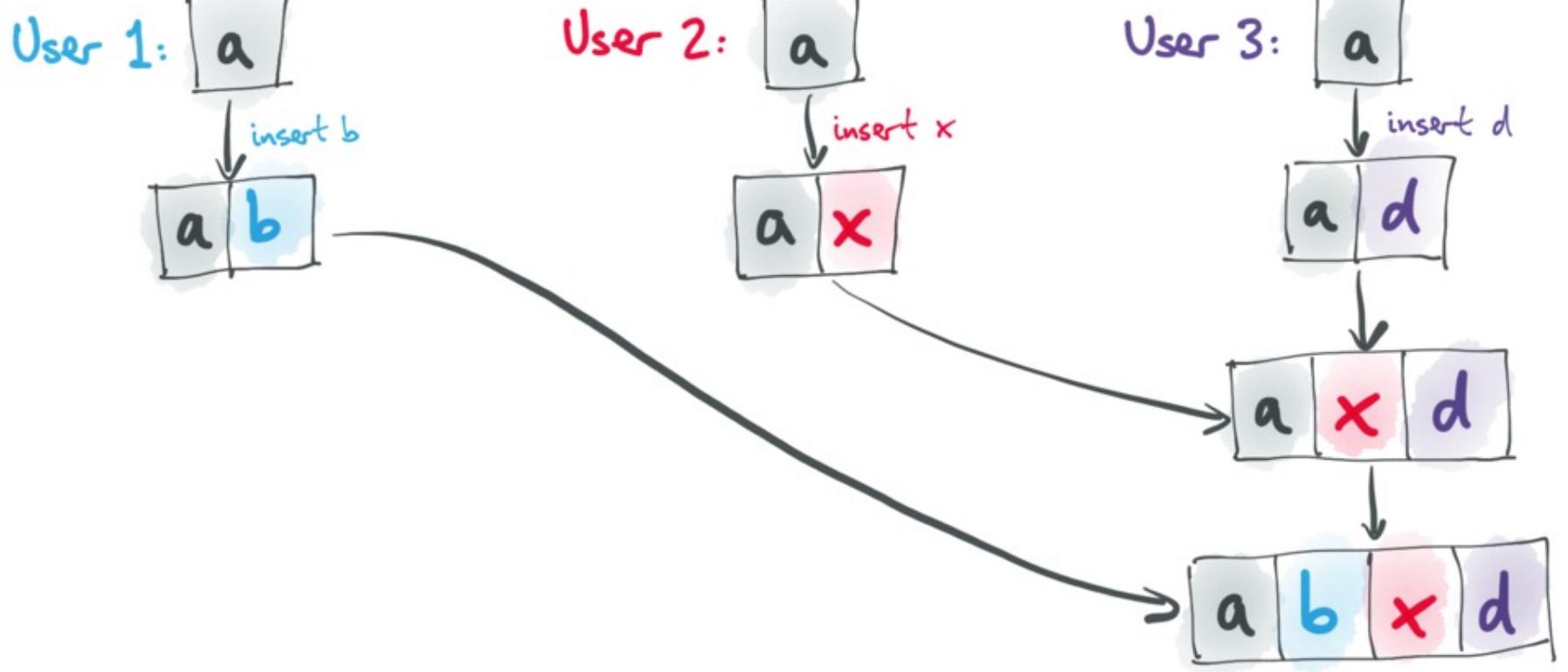
Can we make an algorithm that is non-interleaving
for both forward and backward insertions?



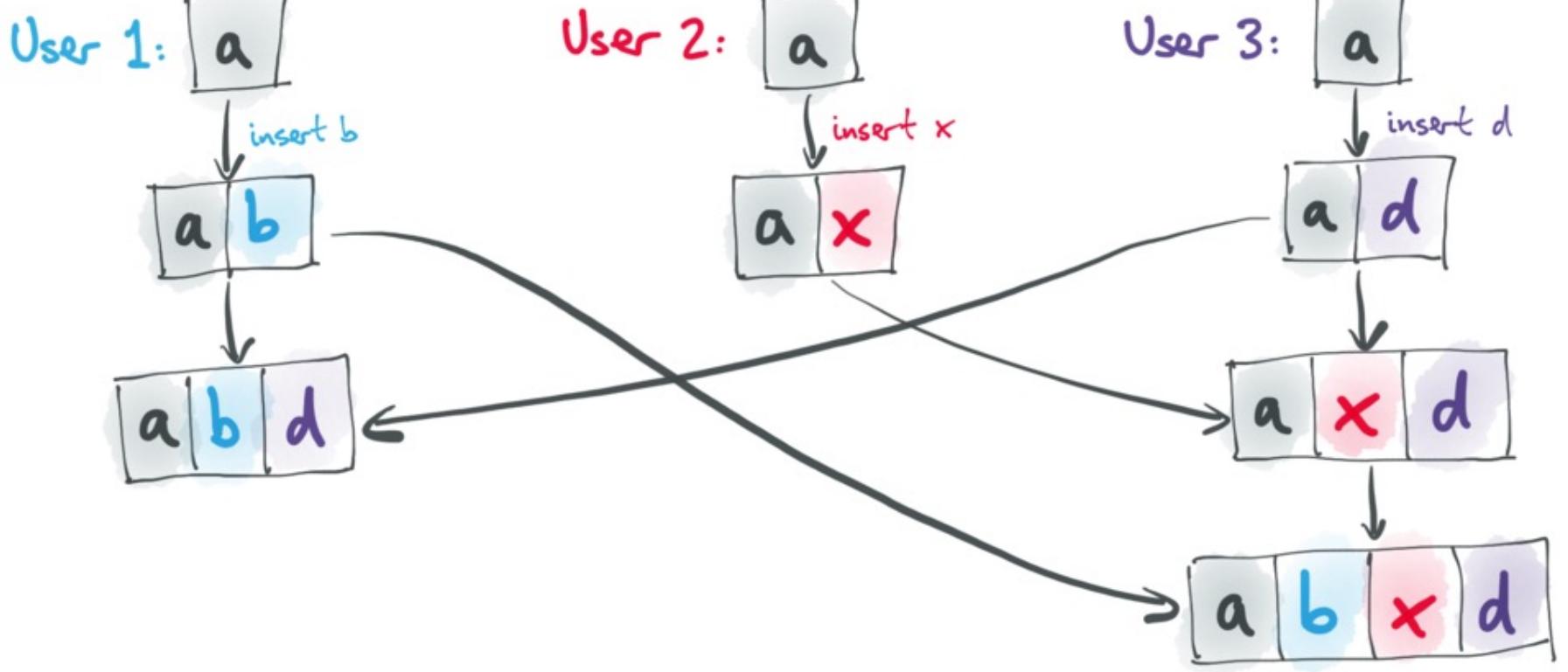
It is impossible for an algorithm to
guarantee both forward and backward
non-interleaving!



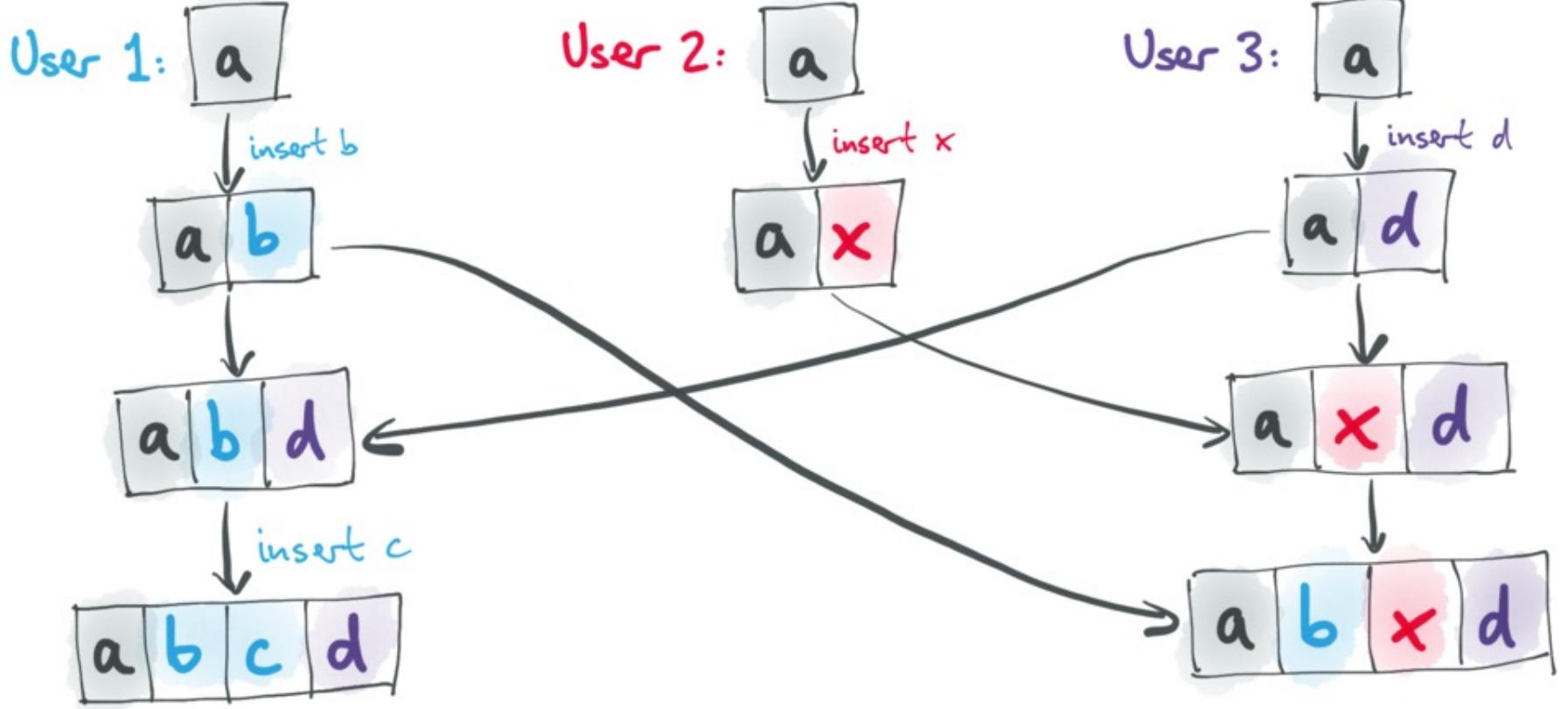




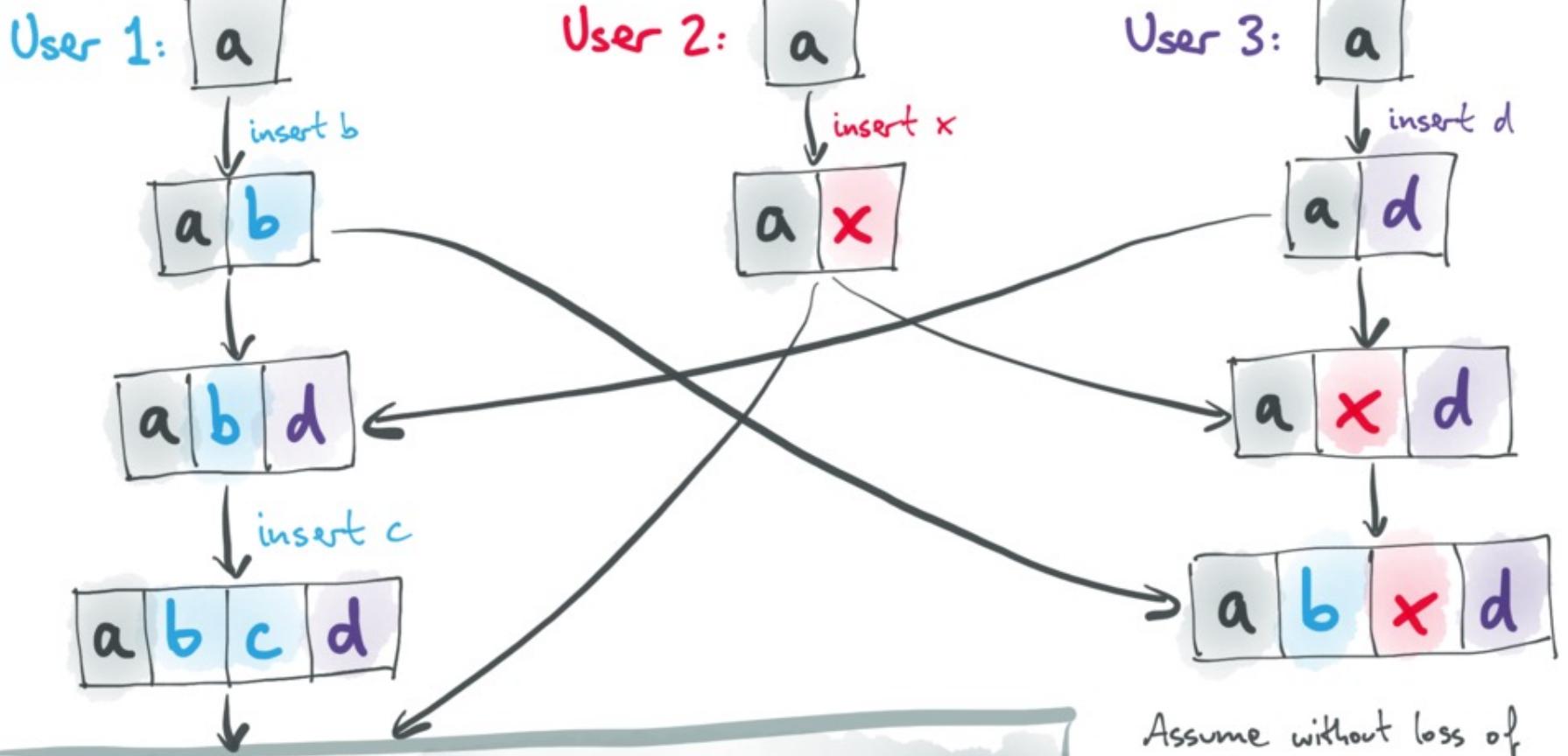
Assume without loss of generality that the three concurrently inserted characters **b, x, d** appear in that order when merged.



Assume without loss of generality that the three concurrently inserted characters **b, x, d** appear in that order when merged.

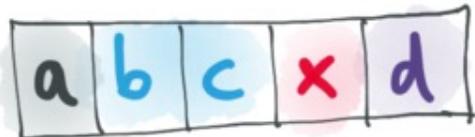
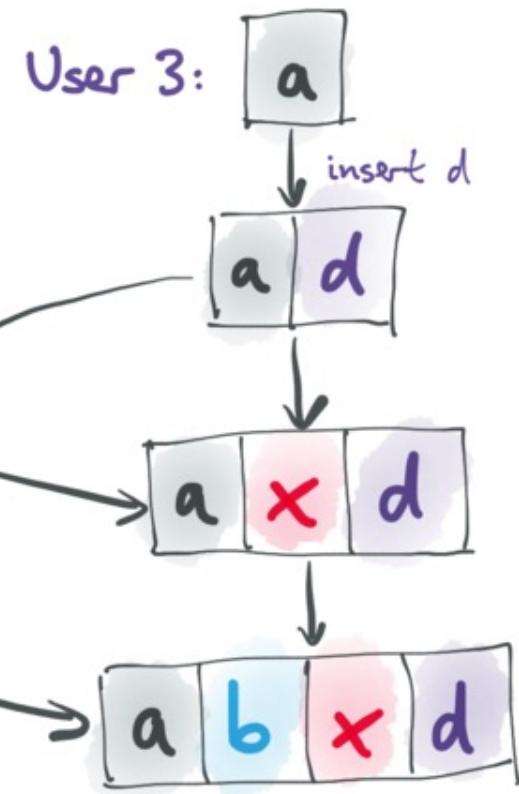
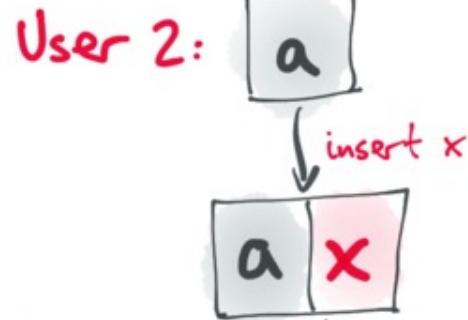
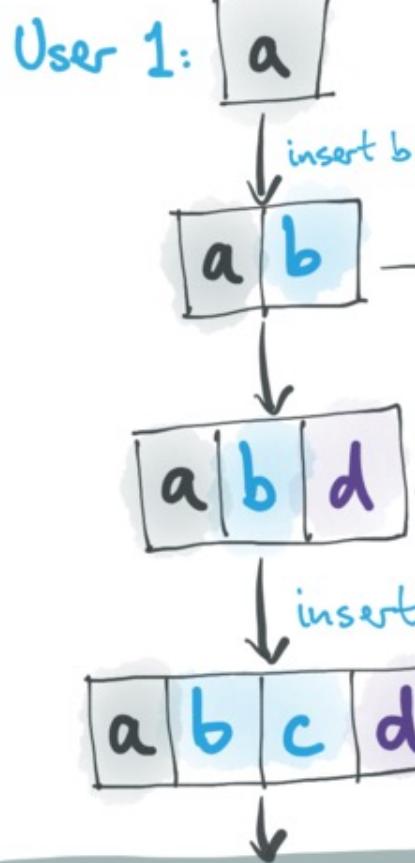


Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.

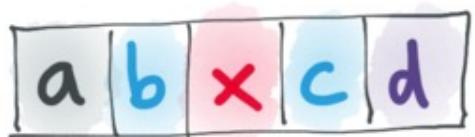


???

Assume without loss of generality that the three concurrently inserted characters **b, x, d** appear in that order when merged.



backward interleaving:
x between c and d



forward interleaving:
x between b and c

Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.

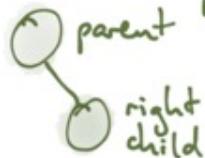
Frugue

A MAXIMALLY NON-INTERLEAVING TEXT CRDT



THE FUGUE ALGORITHM : INTUITION

- A tree where every node is either a right child or a left child



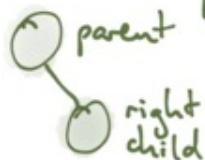
or a left child



(note: not a binary tree – a node may have multiple left and multiple right children)

THE FUGUE ALGORITHM : INTUITION

- A tree where every node is either a right child or a left child



or a left child



(note: not a binary tree – a node may have multiple left and multiple right children)

- Depth-first in-order traversal

(first visit all left subtrees, then the node itself, then all right subtrees.
visit same-side siblings in ID/timestamp order.)

THE FUGUE ALGORITHM : INTUITION

- A tree where every node is either a right child or a left child



or a left child



(note: not a binary tree – a node may have multiple left and multiple right children)

- Depth-first in-order traversal

(first visit all left subtrees, then the node itself, then all right subtrees.
visit same-side siblings in ID/timestamp order.)

- To insert at index i :

- if character at index $i-1$ has no right children, make the new character a right child of the character at index $i-1$
- otherwise, make new character a left child of character at index i

References

- M. Weidner, J. Gentle, M. Kleppmann. The Art of the Fugue: Minimizing Interleaving in Collaborative Text Editing. Preprint, 2023. <https://arxiv.org/abs/2305.00583>
- K.R. McKelvey, S. Jenson, E. Wagner, B. Cook, M. Kleppmann. Upwelling: Combining real-time collaboration with version control for writers. Ink & Switch, 2023. <https://www.inkandswitch.com/upwelling/>
- G. Litt, S. Lim, M. Kleppmann, P. van Hardenberg. Peritext: A CRDT for Collaborative Rich Text Editing. CSCW 2022. doi:10.1145/3555644
- M. Kleppmann, A. Wiggins, P. van Hardenberg, M. McGranaghan. Local-first software: You own your data, in spite of the cloud. Onward! 2019. doi:10.1145/3359591.3359737
- G. Oster, P. Urso, P. Molli, A. Imine. Proving correctness of transformation functions in collaborative editing systems. INRIA Tech Report, RR-5795, 2005. <https://hal.inria.fr/inria-00071213/>
- M. Ressel, D. Nitsche-Ruhland, R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. CSCW 1996. doi:10.1145/240080.240305
- D.A. Nichols, P. Curtis, M. Dixon, J. Lampert. High-latency, low-bandwidth windowing in the Jupiter collaboration system. UIST 1995. doi:10.1145/215585.215706

Project suggestions

- For SOCT2 we have shown interleaving in an implementation of the algorithms (<https://github.com/ept/insert-interleaving>) but not fully explained how it happens. Can you provide an explanation?
- There are various CRDT implementations besides Automerge (e.g. <https://github.com/yjs/yjs>, <https://github.com/josephg/diamond-types>). Please pick one of these, and explain in detail how their algorithm works (you will need to read the source).
- CRDT implementations differ in their support for version control use cases (e.g. diffing two document versions). Can you prototype branching and diffing with one or more of these libraries mentioned above, test the performance, and report which issues you run into?
- Upwelling demonstrates one possible design for integrating version control into a rich text editor. Can you come up with a design for version control in another type of app, such as a spreadsheet, or presentation software?