

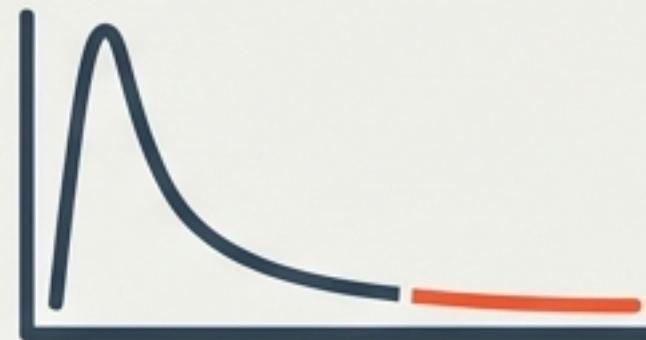
The YouTube Moment for Software

How AI is transforming coding from a technical trade into a mass medium for expression.

```
1 import { insert, Node, NodeList, NodeTree } from 'node-tree';
2 import { NodeList, Node, NodeTree } from 'node-tree';
3 import { NodeList, Node, NodeTree } from 'node-tree';
4 import { NodeList, Node, NodeTree } from 'node-tree';
5
6 class NodeTree {
7   constructor() {
8     this.root = null;
9   }
10
11   // Add a new node to the tree
12   insert(node) {
13     const currentRoot = this.root;
14     const parent = currentRoot;
15     const children = currentRoot.children;
16     const index = currentRoot.children.length;
17
18     if (currentRoot === null) {
19       this.root = node;
20       return;
21     }
22
23     for (let i = 0; i < index; i++) {
24       if (node.value < currentRoot.value) {
25         parent = currentRoot;
26         currentRoot = currentRoot.left;
27       } else if (node.value > currentRoot.value) {
28         parent = currentRoot;
29         currentRoot = currentRoot.right;
30       } else {
31         parent = currentRoot;
32         currentRoot = currentRoot.left;
33       }
34     }
35
36     if (node.value < currentRoot.value) {
37       parent.left = node;
38     } else if (node.value > currentRoot.value) {
39       parent.right = node;
40     } else {
41       parent.left = node;
42     }
43   }
44
45   // Find a node in the tree
46   search(value) {
47     const currentRoot = this.root;
48
49     if (currentRoot === null) {
50       return null;
51     }
52
53     for (let i = 0; i < currentRoot.children.length; i++) {
54       if (currentRoot.children[i].value === value) {
55         return currentRoot;
56       }
57     }
58
59     return null;
60   }
61
62   // Find the minimum value in the tree
63   findMin() {
64     const currentRoot = this.root;
65
66     if (currentRoot === null) {
67       return null;
68     }
69
70     while (currentRoot.left !== null) {
71       currentRoot = currentRoot.left;
72     }
73
74     return currentRoot;
75   }
76
77   // Find the maximum value in the tree
78   findMax() {
79     const currentRoot = this.root;
80
81     if (currentRoot === null) {
82       return null;
83     }
84
85     while (currentRoot.right !== null) {
86       currentRoot = currentRoot.right;
87     }
88
89     return currentRoot;
90   }
91
92   // Print the tree in-order
93   printInOrder() {
94     const currentRoot = this.root;
95
96     if (currentRoot === null) {
97       return;
98     }
99
100    currentRoot.printInOrder();
101    currentRoot.left.printInOrder();
102    currentRoot.right.printInOrder();
103  }
104
105  // Print the tree pre-order
106  printPreOrder() {
107    const currentRoot = this.root;
108
109    if (currentRoot === null) {
110      return;
111    }
112
113    currentRoot.printPreOrder();
114    currentRoot.left.printPreOrder();
115    currentRoot.right.printPreOrder();
116  }
117
118  // Print the tree post-order
119  printPostOrder() {
120    const currentRoot = this.root;
121
122    if (currentRoot === null) {
123      return;
124    }
125
126    currentRoot.left.printPostOrder();
127    currentRoot.right.printPostOrder();
128    currentRoot.printPostOrder();
129  }
130
131  // Print the tree level-order
132  printLevelOrder() {
133    const currentRoot = this.root;
134
135    if (currentRoot === null) {
136      return;
137    }
138
139    const queue = [currentRoot];
140
141    while (queue.length > 0) {
142      const node = queue.shift();
143
144      if (node !== null) {
145        console.log(node.value);
146
147        if (node.left !== null) {
148          queue.push(node.left);
149        }
150
151        if (node.right !== null) {
152          queue.push(node.right);
153        }
154      }
155    }
156  }
157
158  // Print the tree level-order
159  printLevelOrder() {
160    const currentRoot = this.root;
161
162    if (currentRoot === null) {
163      return;
164    }
165
166    const queue = [currentRoot];
167
168    while (queue.length > 0) {
169      const node = queue.shift();
170
171      if (node !== null) {
172        console.log(node.value);
173
174        if (node.left !== null) {
175          queue.push(node.left);
176        }
177
178        if (node.right !== null) {
179          queue.push(node.right);
180        }
181      }
182    }
183  }
184
185  // Print the tree level-order
186  printLevelOrder() {
187    const currentRoot = this.root;
188
189    if (currentRoot === null) {
190      return;
191    }
192
193    const queue = [currentRoot];
194
195    while (queue.length > 0) {
196      const node = queue.shift();
197
198      if (node !== null) {
199        console.log(node.value);
200
201        if (node.left !== null) {
202          queue.push(node.left);
203        }
204
205        if (node.right !== null) {
206          queue.push(node.right);
207        }
208      }
209    }
210  }
211
212  // Print the tree level-order
213  printLevelOrder() {
214    const currentRoot = this.root;
215
216    if (currentRoot === null) {
217      return;
218    }
219
220    const queue = [currentRoot];
221
222    while (queue.length > 0) {
223      const node = queue.shift();
224
225      if (node !== null) {
226        console.log(node.value);
227
228        if (node.left !== null) {
229          queue.push(node.left);
230        }
231
232        if (node.right !== null) {
233          queue.push(node.right);
234        }
235      }
236    }
237  }
238
239  // Print the tree level-order
240  printLevelOrder() {
241    const currentRoot = this.root;
242
243    if (currentRoot === null) {
244      return;
245    }
246
247    const queue = [currentRoot];
248
249    while (queue.length > 0) {
250      const node = queue.shift();
251
252      if (node !== null) {
253        console.log(node.value);
254
255        if (node.left !== null) {
256          queue.push(node.left);
257        }
258
259        if (node.right !== null) {
260          queue.push(node.right);
261        }
262      }
263    }
264  }
265
266  // Print the tree level-order
267  printLevelOrder() {
268    const currentRoot = this.root;
269
270    if (currentRoot === null) {
271      return;
272    }
273
274    const queue = [currentRoot];
275
276    while (queue.length > 0) {
277      const node = queue.shift();
278
279      if (node !== null) {
280        console.log(node.value);
281
282        if (node.left !== null) {
283          queue.push(node.left);
284        }
285
286        if (node.right !== null) {
287          queue.push(node.right);
288        }
289      }
290    }
291  }
292
293  // Print the tree level-order
294  printLevelOrder() {
295    const currentRoot = this.root;
296
297    if (currentRoot === null) {
298      return;
299    }
300
301    const queue = [currentRoot];
302
303    while (queue.length > 0) {
304      const node = queue.shift();
305
306      if (node !== null) {
307        console.log(node.value);
308
309        if (node.left !== null) {
310          queue.push(node.left);
311        }
312
313        if (node.right !== null) {
314          queue.push(node.right);
315        }
316      }
317    }
318  }
319
320  // Print the tree level-order
321  printLevelOrder() {
322    const currentRoot = this.root;
323
324    if (currentRoot === null) {
325      return;
326    }
327
328    const queue = [currentRoot];
329
330    while (queue.length > 0) {
331      const node = queue.shift();
332
333      if (node !== null) {
334        console.log(node.value);
335
336        if (node.left !== null) {
337          queue.push(node.left);
338        }
339
340        if (node.right !== null) {
341          queue.push(node.right);
342        }
343      }
344    }
345  }
346
347  // Print the tree level-order
348  printLevelOrder() {
349    const currentRoot = this.root;
350
351    if (currentRoot === null) {
352      return;
353    }
354
355    const queue = [currentRoot];
356
357    while (queue.length > 0) {
358      const node = queue.shift();
359
360      if (node !== null) {
361        console.log(node.value);
362
363        if (node.left !== null) {
364          queue.push(node.left);
365        }
366
367        if (node.right !== null) {
368          queue.push(node.right);
369        }
370      }
371    }
372  }
373
374  // Print the tree level-order
375  printLevelOrder() {
376    const currentRoot = this.root;
377
378    if (currentRoot === null) {
379      return;
380    }
381
382    const queue = [currentRoot];
383
384    while (queue.length > 0) {
385      const node = queue.shift();
386
387      if (node !== null) {
388        console.log(node.value);
389
390        if (node.left !== null) {
391          queue.push(node.left);
392        }
393
394        if (node.right !== null) {
395          queue.push(node.right);
396        }
397      }
398    }
399  }
400
401  // Print the tree level-order
402  printLevelOrder() {
403    const currentRoot = this.root;
404
405    if (currentRoot === null) {
406      return;
407    }
408
409    const queue = [currentRoot];
410
411    while (queue.length > 0) {
412      const node = queue.shift();
413
414      if (node !== null) {
415        console.log(node.value);
416
417        if (node.left !== null) {
418          queue.push(node.left);
419        }
420
421        if (node.right !== null) {
422          queue.push(node.right);
423        }
424      }
425    }
426  }
427
428  // Print the tree level-order
429  printLevelOrder() {
430    const currentRoot = this.root;
431
432    if (currentRoot === null) {
433      return;
434    }
435
436    const queue = [currentRoot];
437
438    while (queue.length > 0) {
439      const node = queue.shift();
440
441      if (node !== null) {
442        console.log(node.value);
443
444        if (node.left !== null) {
445          queue.push(node.left);
446        }
447
448        if (node.right !== null) {
449          queue.push(node.right);
450        }
451      }
452    }
453  }
454
455  // Print the tree level-order
456  printLevelOrder() {
457    const currentRoot = this.root;
458
459    if (currentRoot === null) {
460      return;
461    }
462
463    const queue = [currentRoot];
464
465    while (queue.length > 0) {
466      const node = queue.shift();
467
468      if (node !== null) {
469        console.log(node.value);
470
471        if (node.left !== null) {
472          queue.push(node.left);
473        }
474
475        if (node.right !== null) {
476          queue.push(node.right);
477        }
478      }
479    }
480  }
481
482  // Print the tree level-order
483  printLevelOrder() {
484    const currentRoot = this.root;
485
486    if (currentRoot === null) {
487      return;
488    }
489
490    const queue = [currentRoot];
491
492    while (queue.length > 0) {
493      const node = queue.shift();
494
495      if (node !== null) {
496        console.log(node.value);
497
498        if (node.left !== null) {
499          queue.push(node.left);
500        }
501
502        if (node.right !== null) {
503          queue.push(node.right);
504        }
505      }
506    }
507  }
508
509  // Print the tree level-order
510  printLevelOrder() {
511    const currentRoot = this.root;
512
513    if (currentRoot === null) {
514      return;
515    }
516
517    const queue = [currentRoot];
518
519    while (queue.length > 0) {
520      const node = queue.shift();
521
522      if (node !== null) {
523        console.log(node.value);
524
525        if (node.left !== null) {
526          queue.push(node.left);
527        }
528
529        if (node.right !== null) {
530          queue.push(node.right);
531        }
532      }
533    }
534  }
535
536  // Print the tree level-order
537  printLevelOrder() {
538    const currentRoot = this.root;
539
540    if (currentRoot === null) {
541      return;
542    }
543
544    const queue = [currentRoot];
545
546    while (queue.length > 0) {
547      const node = queue.shift();
548
549      if (node !== null) {
550        console.log(node.value);
551
552        if (node.left !== null) {
553          queue.push(node.left);
554        }
555
556        if (node.right !== null) {
557          queue.push(node.right);
558        }
559      }
560    }
561  }
562
563  // Print the tree level-order
564  printLevelOrder() {
565    const currentRoot = this.root;
566
567    if (currentRoot === null) {
568      return;
569    }
570
571    const queue = [currentRoot];
572
573    while (queue.length > 0) {
574      const node = queue.shift();
575
576      if (node !== null) {
577        console.log(node.value);
578
579        if (node.left !== null) {
580          queue.push(node.left);
581        }
582
583        if (node.right !== null) {
584          queue.push(node.right);
585        }
586      }
587    }
588  }
589
590  // Print the tree level-order
591  printLevelOrder() {
592    const currentRoot = this.root;
593
594    if (currentRoot === null) {
595      return;
596    }
597
598    const queue = [currentRoot];
599
600    while (queue.length > 0) {
601      const node = queue.shift();
602
603      if (node !== null) {
604        console.log(node.value);
605
606        if (node.left !== null) {
607          queue.push(node.left);
608        }
609
610        if (node.right !== null) {
611          queue.push(node.right);
612        }
613      }
614    }
615  }
616
617  // Print the tree level-order
618  printLevelOrder() {
619    const currentRoot = this.root;
620
621    if (currentRoot === null) {
622      return;
623    }
624
625    const queue = [currentRoot];
626
627    while (queue.length > 0) {
628      const node = queue.shift();
629
630      if (node !== null) {
631        console.log(node.value);
632
633        if (node.left !== null) {
634          queue.push(node.left);
635        }
636
637        if (node.right !== null) {
638          queue.push(node.right);
639        }
640      }
641    }
642  }
643
644  // Print the tree level-order
645  printLevelOrder() {
646    const currentRoot = this.root;
647
648    if (currentRoot === null) {
649      return;
650    }
651
652    const queue = [currentRoot];
653
654    while (queue.length > 0) {
655      const node = queue.shift();
656
657      if (node !== null) {
658        console.log(node.value);
659
660        if (node.left !== null) {
661          queue.push(node.left);
662        }
663
664        if (node.right !== null) {
665          queue.push(node.right);
666        }
667      }
668    }
669  }
670
671  // Print the tree level-order
672  printLevelOrder() {
673    const currentRoot = this.root;
674
675    if (currentRoot === null) {
676      return;
677    }
678
679    const queue = [currentRoot];
680
681    while (queue.length > 0) {
682      const node = queue.shift();
683
684      if (node !== null) {
685        console.log(node.value);
686
687        if (node.left !== null) {
688          queue.push(node.left);
689        }
690
691        if (node.right !== null) {
692          queue.push(node.right);
693        }
694      }
695    }
696  }
697
698  // Print the tree level-order
699  printLevelOrder() {
700    const currentRoot = this.root;
701
702    if (currentRoot === null) {
703      return;
704    }
705
706    const queue = [currentRoot];
707
708    while (queue.length > 0) {
709      const node = queue.shift();
710
711      if (node !== null) {
712        console.log(node.value);
713
714        if (node.left !== null) {
715          queue.push(node.left);
716        }
717
718        if (node.right !== null) {
719          queue.push(node.right);
720        }
721      }
722    }
723  }
724
725  // Print the tree level-order
726  printLevelOrder() {
727    const currentRoot = this.root;
728
729    if (currentRoot === null) {
730      return;
731    }
732
733    const queue = [currentRoot];
734
735    while (queue.length > 0) {
736      const node = queue.shift();
737
738      if (node !== null) {
739        console.log(node.value);
740
741        if (node.left !== null) {
742          queue.push(node.left);
743        }
744
745        if (node.right !== null) {
746          queue.push(node.right);
747        }
748      }
749    }
750  }
751
752  // Print the tree level-order
753  printLevelOrder() {
754    const currentRoot = this.root;
755
756    if (currentRoot === null) {
757      return;
758    }
759
760    const queue = [currentRoot];
761
762    while (queue.length > 0) {
763      const node = queue.shift();
764
765      if (node !== null) {
766        console.log(node.value);
767
768        if (node.left !== null) {
769          queue.push(node.left);
770        }
771
772        if (node.right !== null) {
773          queue.push(node.right);
774        }
775      }
776    }
777  }
778
779  // Print the tree level-order
780  printLevelOrder() {
781    const currentRoot = this.root;
782
783    if (currentRoot === null) {
784      return;
785    }
786
787    const queue = [currentRoot];
788
789    while (queue.length > 0) {
790      const node = queue.shift();
791
792      if (node !== null) {
793        console.log(node.value);
794
795        if (node.left !== null) {
796          queue.push(node.left);
797        }
798
799        if (node.right !== null) {
800          queue.push(node.right);
801        }
802      }
803    }
804  }
805
806  // Print the tree level-order
807  printLevelOrder() {
808    const currentRoot = this.root;
809
810    if (currentRoot === null) {
811      return;
812    }
813
814    const queue = [currentRoot];
815
816    while (queue.length > 0) {
817      const node = queue.shift();
818
819      if (node !== null) {
820        console.log(node.value);
821
822        if (node.left !== null) {
823          queue.push(node.left);
824        }
825
826        if (node.right !== null) {
827          queue.push(node.right);
828        }
829      }
830    }
831  }
832
833  // Print the tree level-order
834  printLevelOrder() {
835    const currentRoot = this.root;
836
837    if (currentRoot === null) {
838      return;
839    }
840
841    const queue = [currentRoot];
842
843    while (queue.length > 0) {
844      const node = queue.shift();
845
846      if (node !== null) {
847        console.log(node.value);
848
849        if (node.left !== null) {
850          queue.push(node.left);
851        }
852
853        if (node.right !== null) {
854          queue.push(node.right);
855        }
856      }
857    }
858  }
859
860  // Print the tree level-order
861  printLevelOrder() {
862    const currentRoot = this.root;
863
864    if (currentRoot === null) {
865      return;
866    }
867
868    const queue = [currentRoot];
869
870    while (queue.length > 0) {
871      const node = queue.shift();
872
873      if (node !== null) {
874        console.log(node.value);
875
876        if (node.left !== null) {
877          queue.push(node.left);
878        }
879
880        if (node.right !== null) {
881          queue.push(node.right);
882        }
883      }
884    }
885  }
886
887  // Print the tree level-order
888  printLevelOrder() {
889    const currentRoot = this.root;
890
891    if (currentRoot === null) {
892      return;
893    }
894
895    const queue = [currentRoot];
896
897    while (queue.length > 0) {
898      const node = queue.shift();
899
900      if (node !== null) {
901        console.log(node.value);
902
903        if (node.left !== null) {
904          queue.push(node.left);
905        }
906
907        if (node.right !== null) {
908          queue.push(node.right);
909        }
910      }
911    }
912  }
913
914  // Print the tree level-order
915  printLevelOrder() {
916    const currentRoot = this.root;
917
918    if (currentRoot === null) {
919      return;
920    }
921
922    const queue = [currentRoot];
923
924    while (queue.length > 0) {
925      const node = queue.shift();
926
927      if (node !== null) {
928        console.log(node.value);
929
930        if (node.left !== null) {
931          queue.push(node.left);
932        }
933
934        if (node.right !== null) {
935          queue.push(node.right);
936        }
937      }
938    }
939  }
940
941  // Print the tree level-order
942  printLevelOrder() {
943    const currentRoot = this.root;
944
945    if (currentRoot === null) {
946      return;
947    }
948
949    const queue = [currentRoot];
950
951    while (queue.length > 0) {
952      const node = queue.shift();
953
954      if (node !== null) {
955        console.log(node.value);
956
957        if (node.left !== null) {
958          queue.push(node.left);
959        }
960
961        if (node.right !== null) {
962          queue.push(node.right);
963        }
964      }
965    }
966  }
967
968  // Print the tree level-order
969  printLevelOrder() {
970    const currentRoot = this.root;
971
972    if (currentRoot === null) {
973      return;
974    }
975
976    const queue = [currentRoot];
977
978    while (queue.length > 0) {
979      const node = queue.shift();
980
981      if (node !== null) {
982        console.log(node.value);
983
984        if (node.left !== null) {
985          queue.push(node.left);
986        }
987
988        if (node.right !== null) {
989          queue.push(node.right);
990        }
991      }
992    }
993  }
994
995  // Print the tree level-order
996  printLevelOrder() {
997    const currentRoot = this.root;
998
999    if (currentRoot === null) {
1000      return;
1001    }
1002
1003    const queue = [currentRoot];
1004
1005    while (queue.length &
```

Coding is entering its mass adoption phase

THE TREND



Software creation is undergoing a 'long-tail' expansion distinctively similar to video in 2005. Niche, personal, and single-use software is now viable.

THE DRIVER



AI tools have compressed development time from weeks to hours, effectively removing the 'syntax barrier' that kept non-engineers out.

THE SHIFT



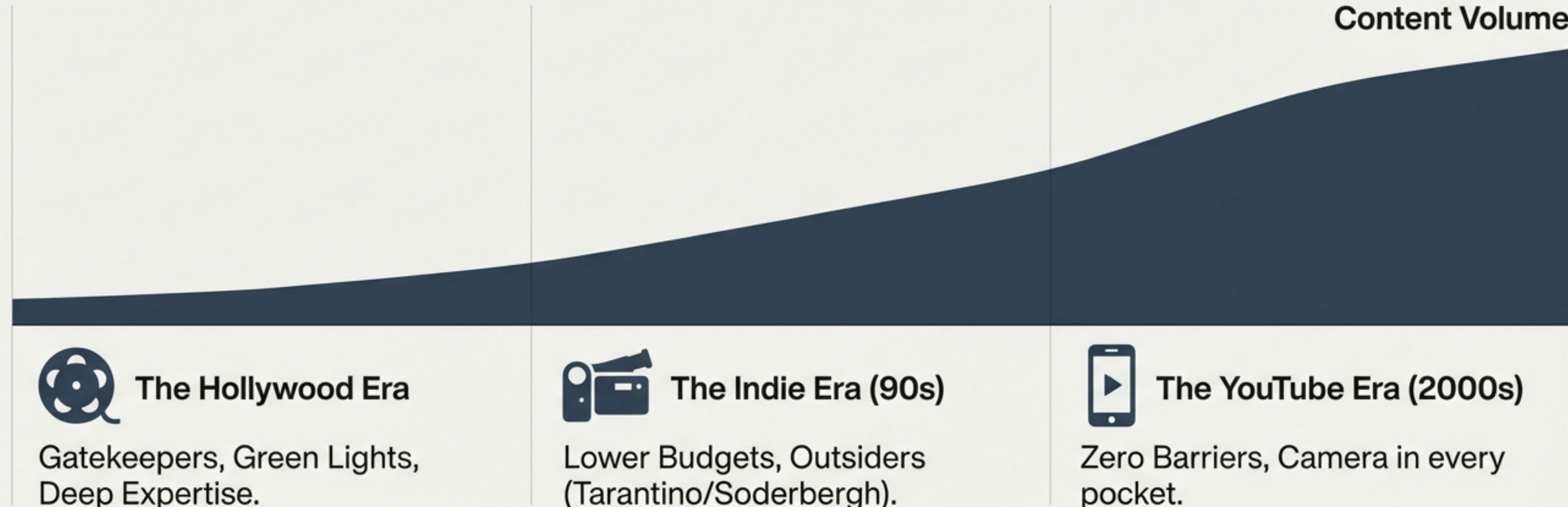
Software is moving from strictly 'Utility' (solving business problems) to 'Expression' (creative, funny, niche, cultural).

THE OPPORTUNITY



Unlike video content which decays rapidly, software accrues value over time. We are entering the era of the 'citizen developer'.

History doesn't repeat, but the mechanics match



Insight: In 2005, YouTube didn't seem to fill an obvious gap. Two decades later, it is a \$550 billion business more culturally relevant than traditional TV.

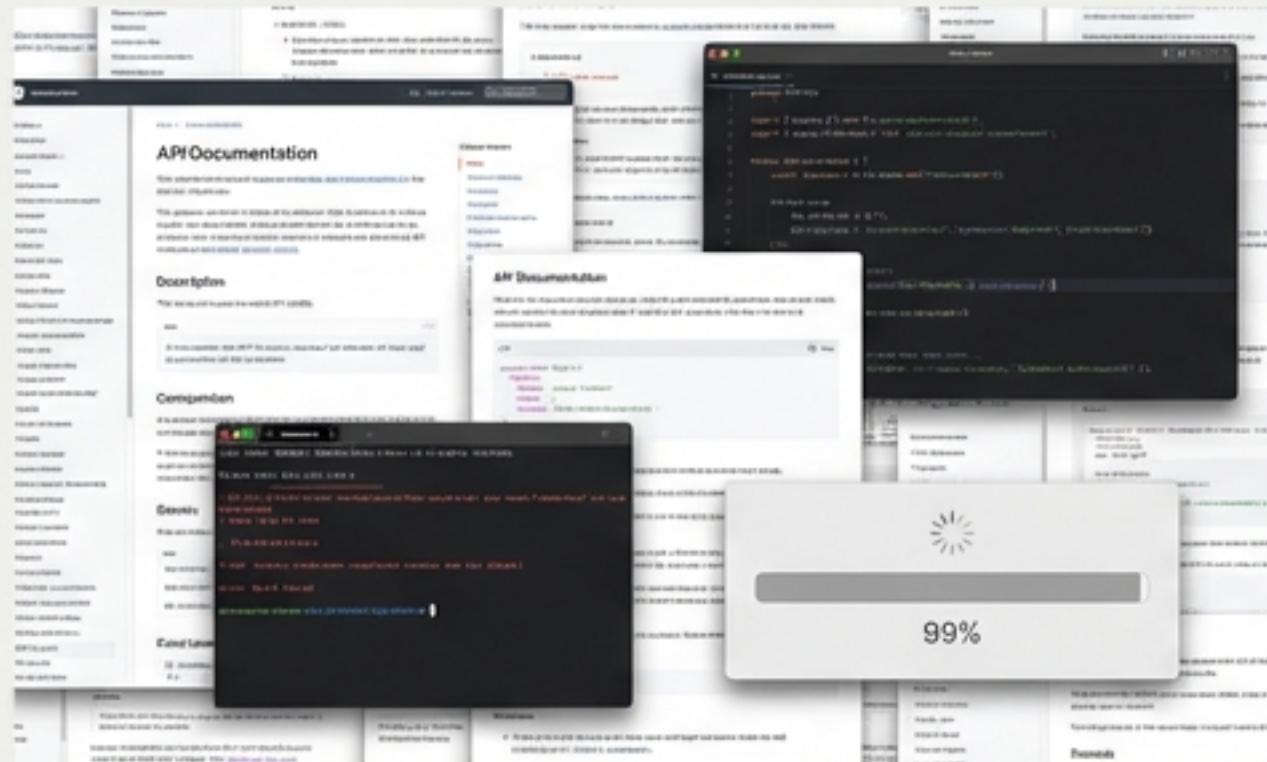
We are leaving the ‘Hollywood Era’ of Code



If you previously had excuses for not building something, they just got a lot less convincing.

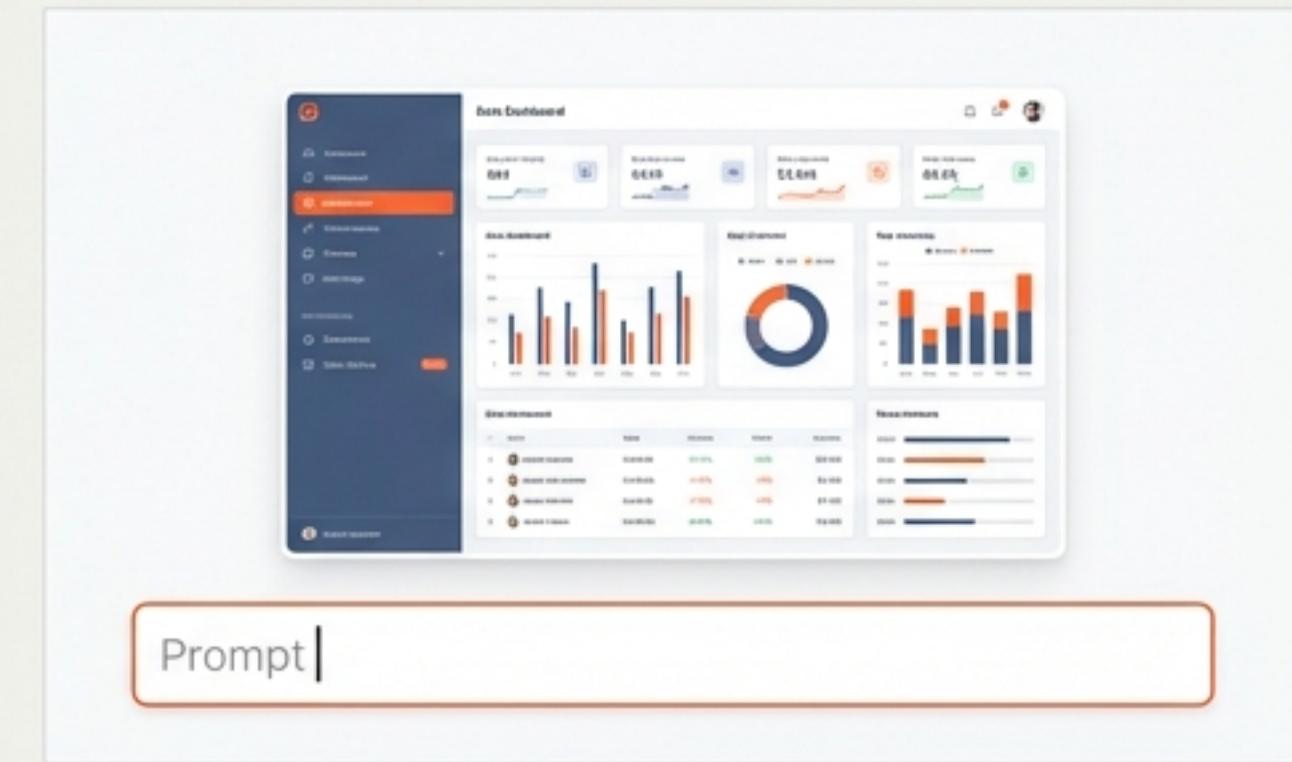
The collapse of the barrier to entry

THE OLD WAY (Weeks)



- Learn Syntax
- Dig through API Docs
- Configure Localhost
- Maintain Frameworks

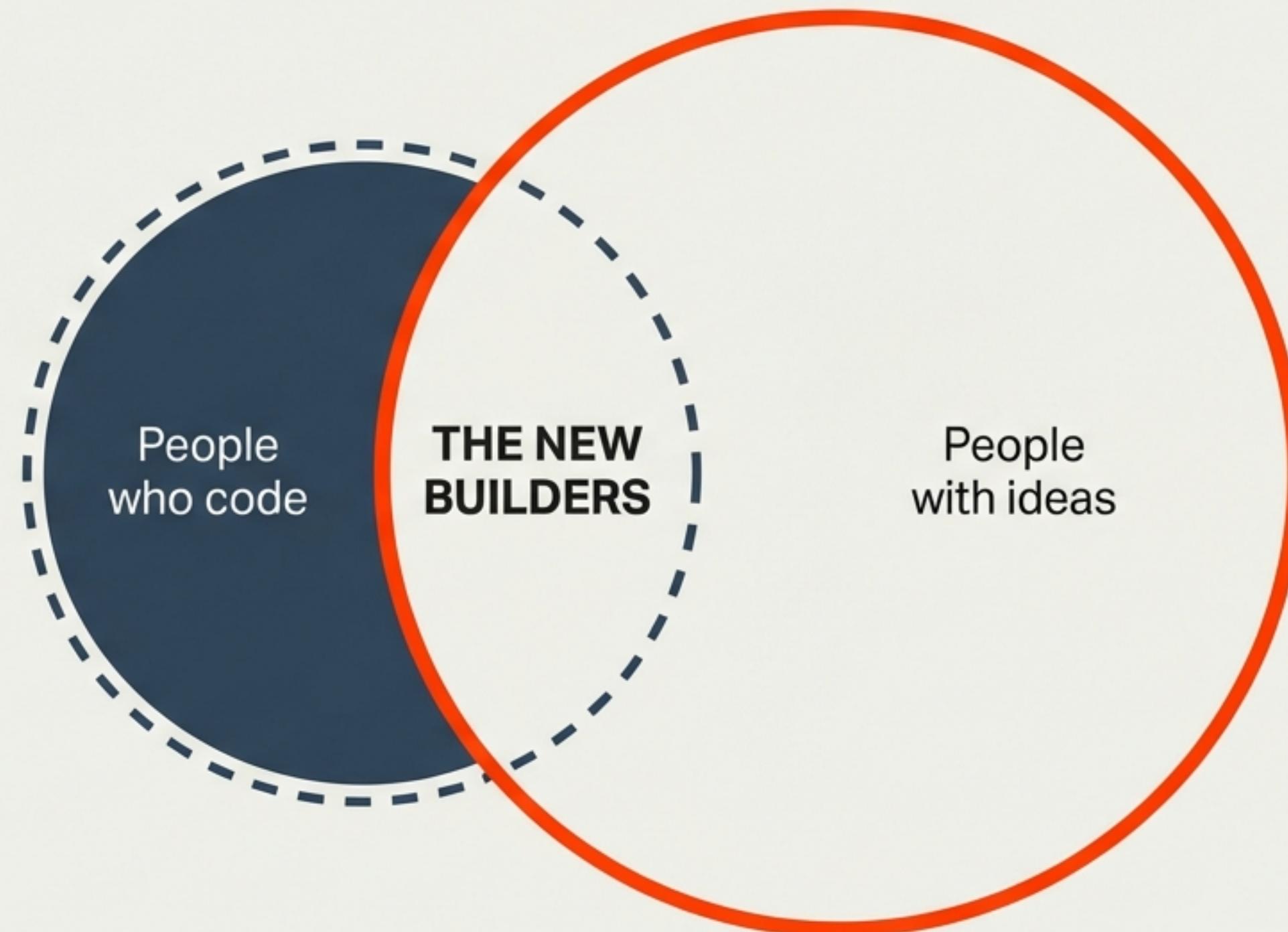
THE NEW WAY (Hours)



- Type a text prompt
- Ship with Cursor / Wabi / Replit
- Zero Friction

Evidence: Tobi's custom MRI dashboard and Marc's movie recommender—tools that previously required commercial teams, now built by individuals.

You no longer need to care about software to build it



Previous State: Builders were a niche subculture—Tech Twitter, Paul Graham readers.

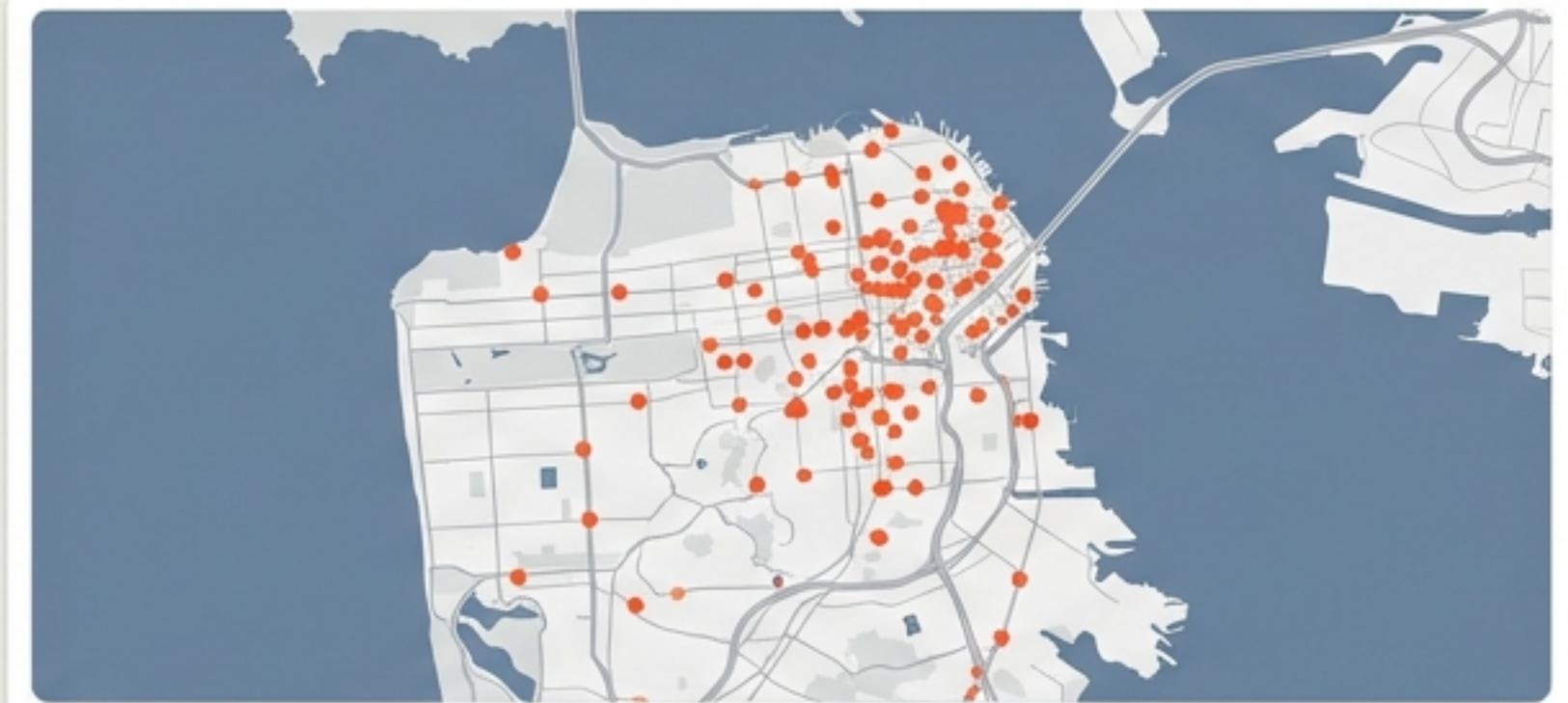
Current State: The addressable market is anyone with a good idea.

Insight: Democratization means the user becomes the maker. Doctors, teachers, and shop owners are finally designing products for problems they actually live.

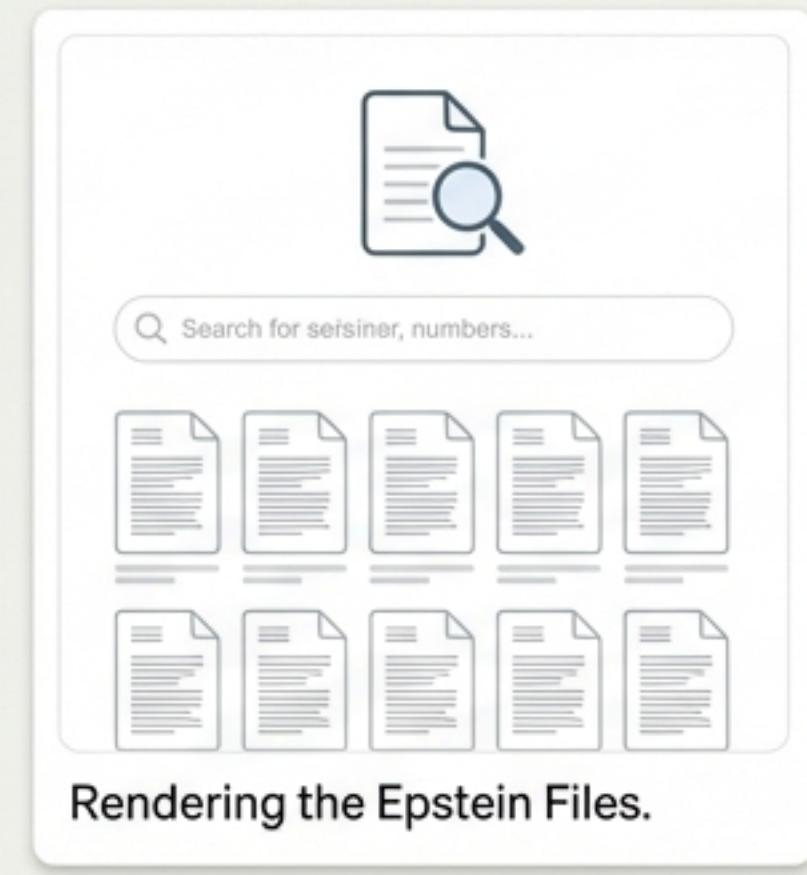
Software is becoming a medium for expression

Moving beyond Utility. Software is becoming a vehicle for personality, humor, and zeitgeist—just like a tweet or a video.

The Shift: People will soon create funny software with about as much effort as drafting a funny post on X. We are witnessing the rise of the “Tiny App”.



Riley Walz: Visualizing SF Parking Cops.



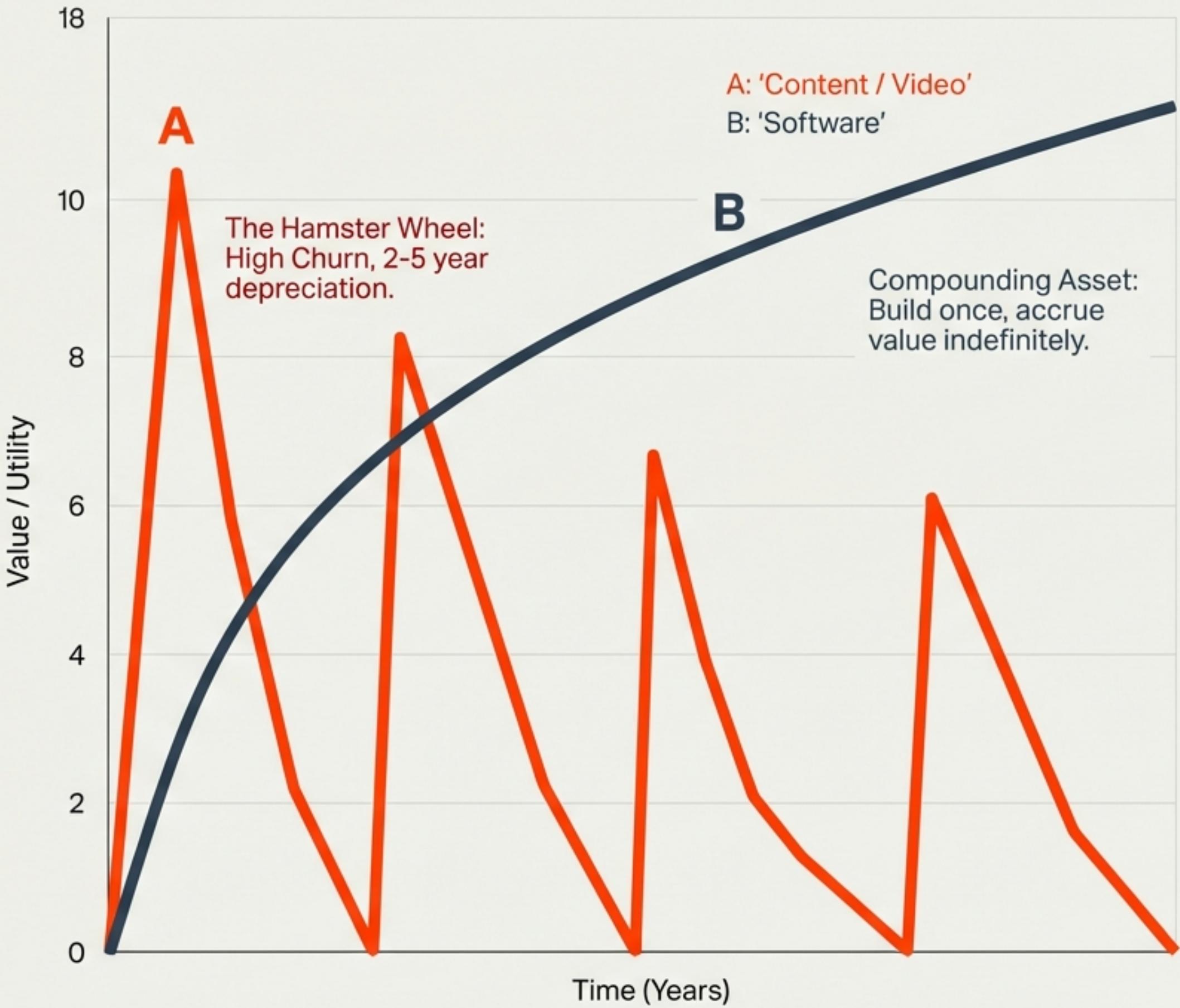
Rendering the Epstein Files.



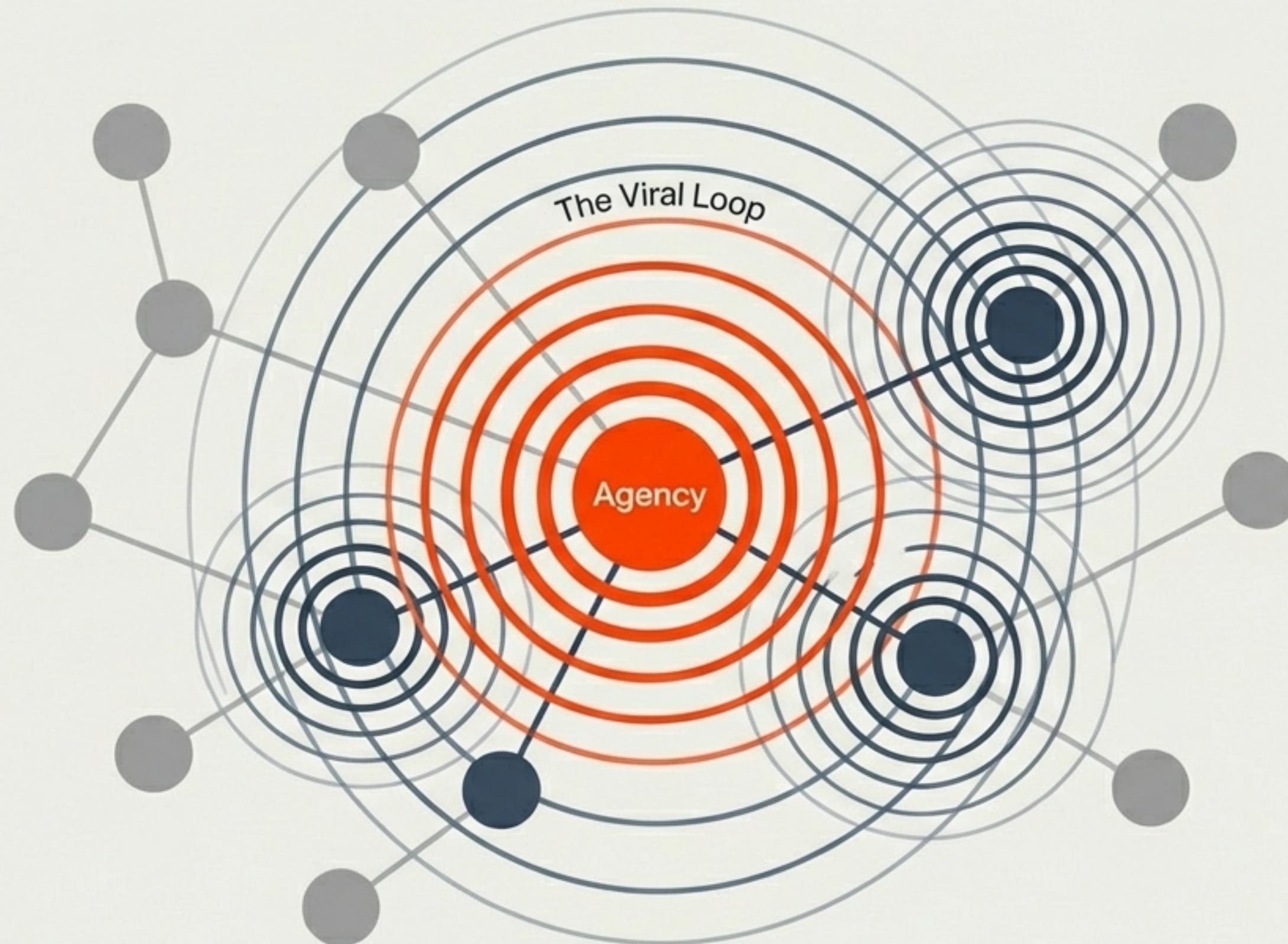
Niche Humor App.

Why Software is a superior asset class to Content

Tiny, expressive software 'posts' can evolve into durable businesses. This is the underappreciated advantage of the medium.



Mimetic Desire & The New ‘Cool’



Why are people building?
Because they see their friends
doing it.

1. The Viral Loop: Seeing a
friend ship an app creates the
'I can do that too' impulse.

2. Cultural Shift: It's no longer
about aspiring to be an
'Influencer' for fame; it's
about Agency—taking
destiny into your own hands.

There is no angst, only envy.

There has never been a better
time to be a young person
with great ideas.

The Last Mile Problem: Distribution



We have moved from ‘Everyone can build’ to ‘Everyone can try,’ but we have not yet reached “Everyone can distribute.” The ecosystem is waiting for the true “YouTube Platform”—where publishing is as simple as uploading.

The Kids Are Alright

AI has democratized leverage. Tiny, expressive software “posts” will evolve into durable businesses. The barriers are gone.

We are witnessing the birth of a new viral medium where “software” is the message.

**The question isn't 'can you build it?'
It's 'why haven't you started?'**