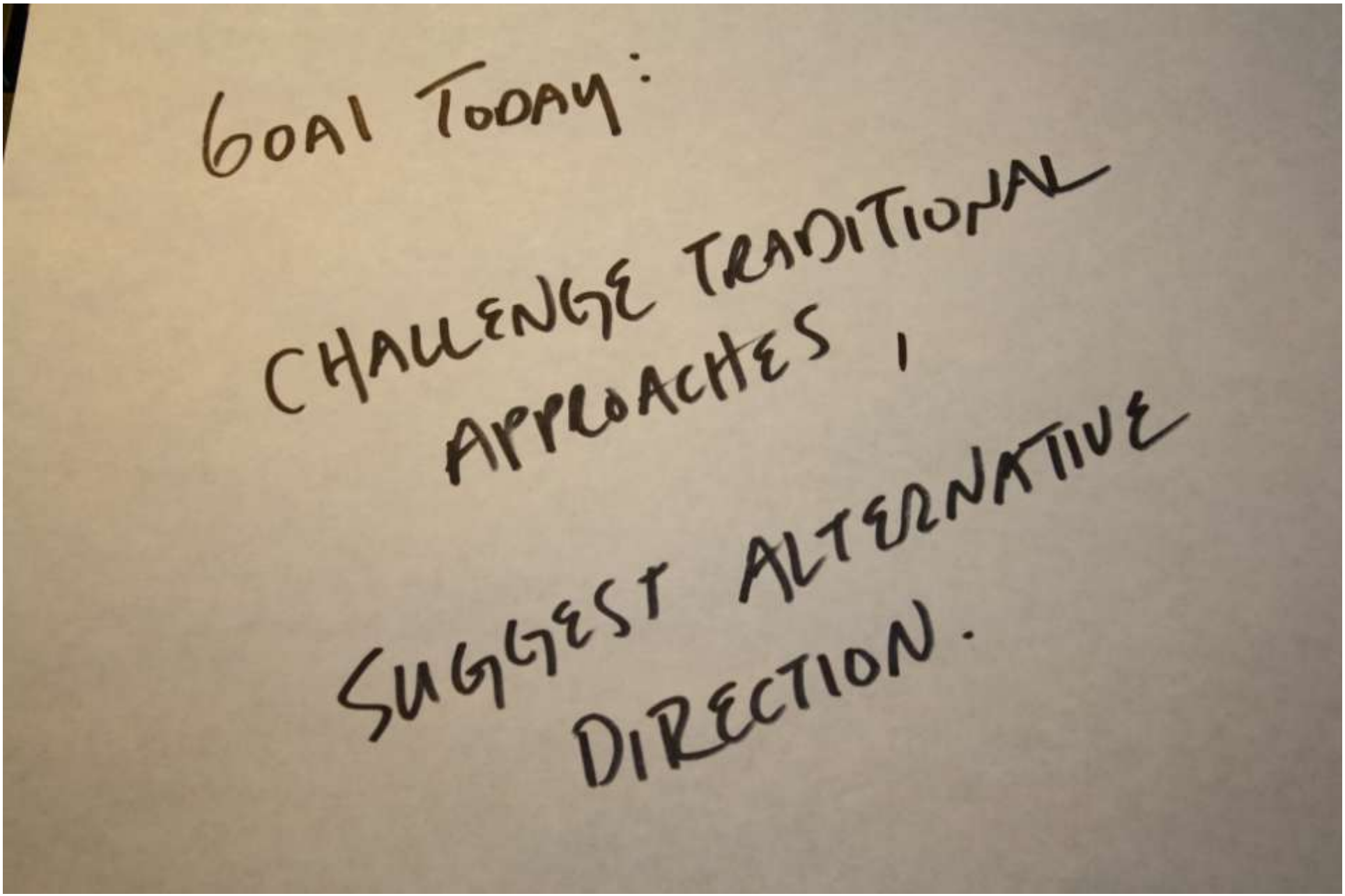
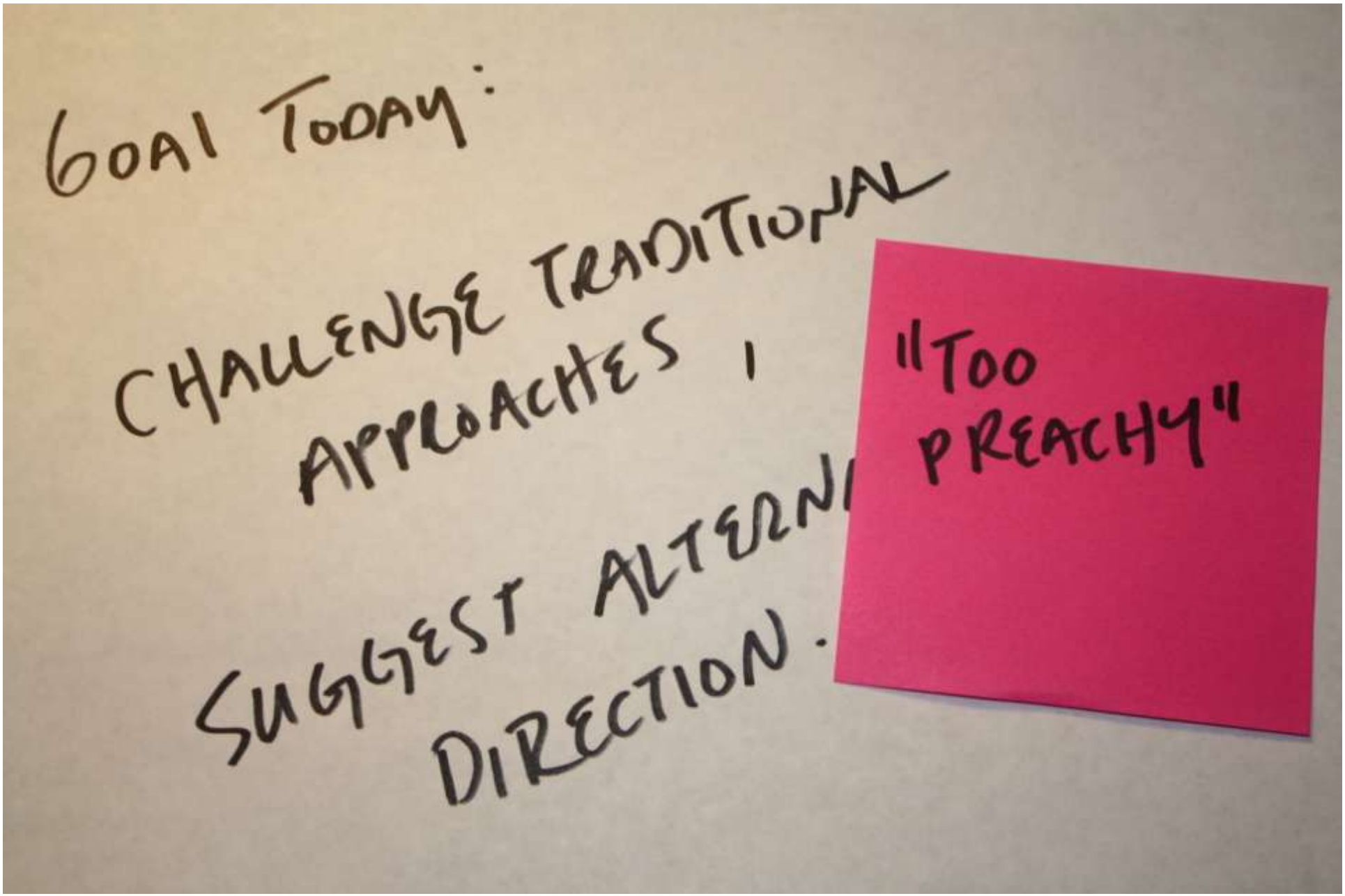
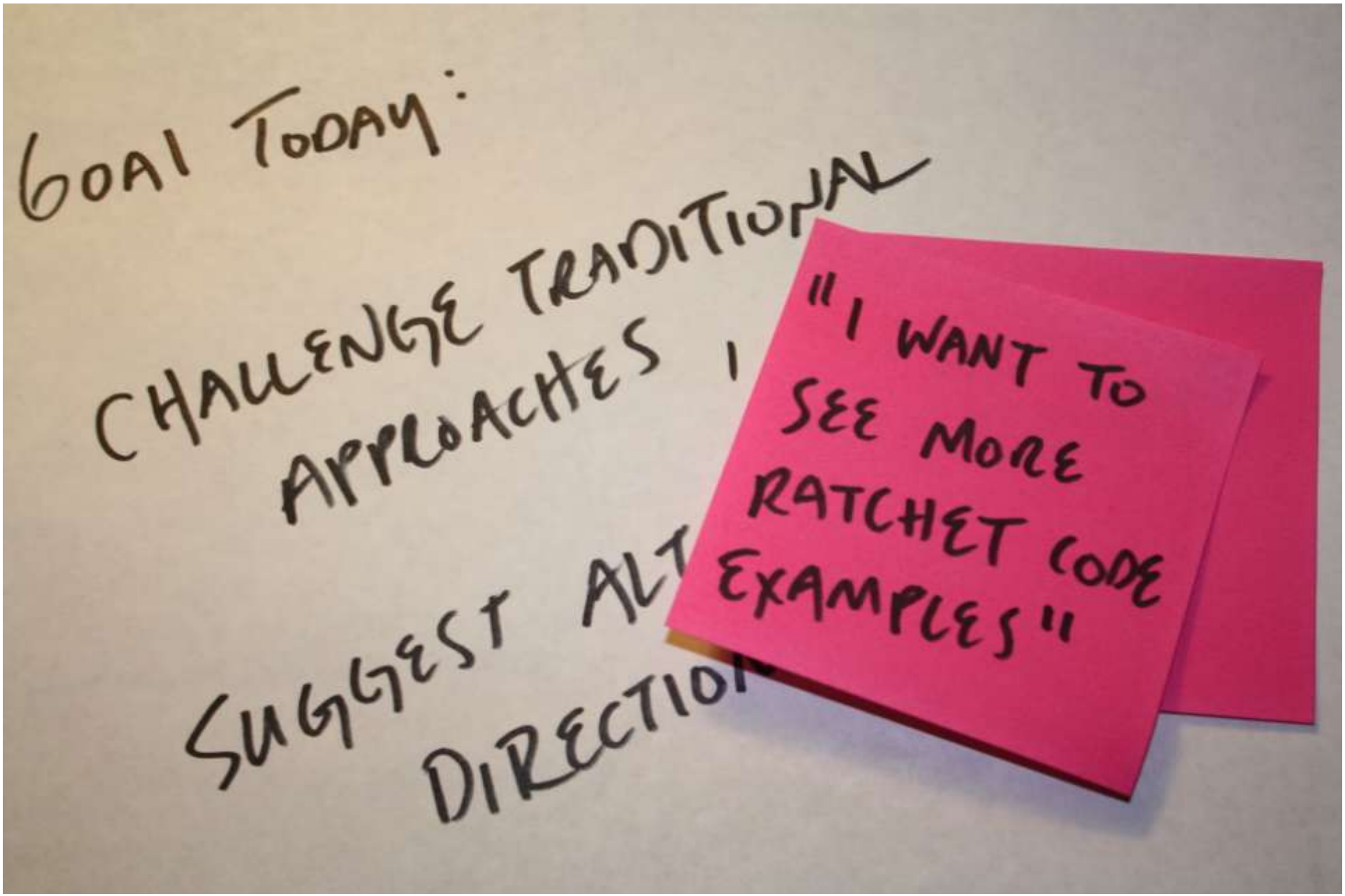
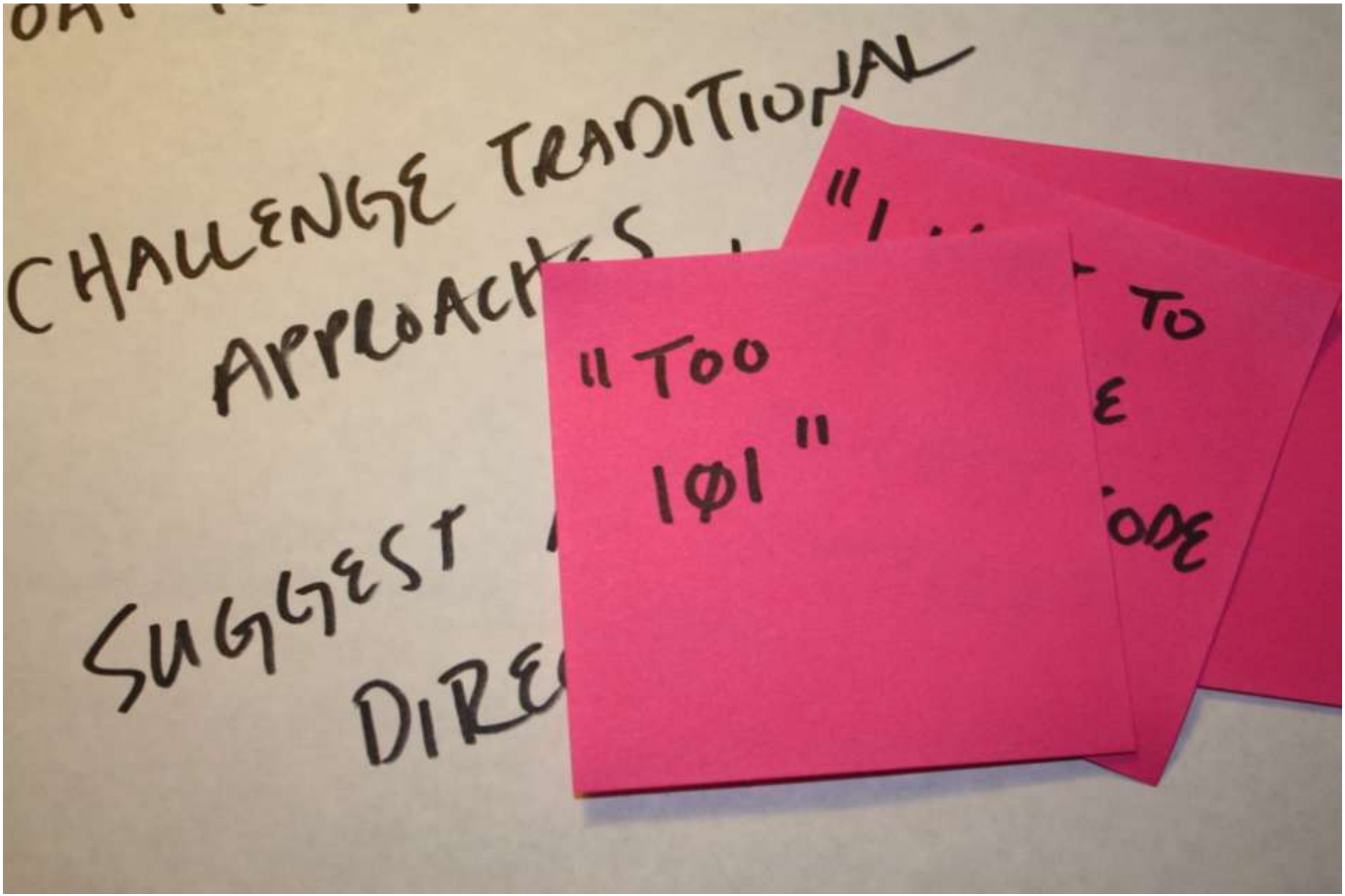
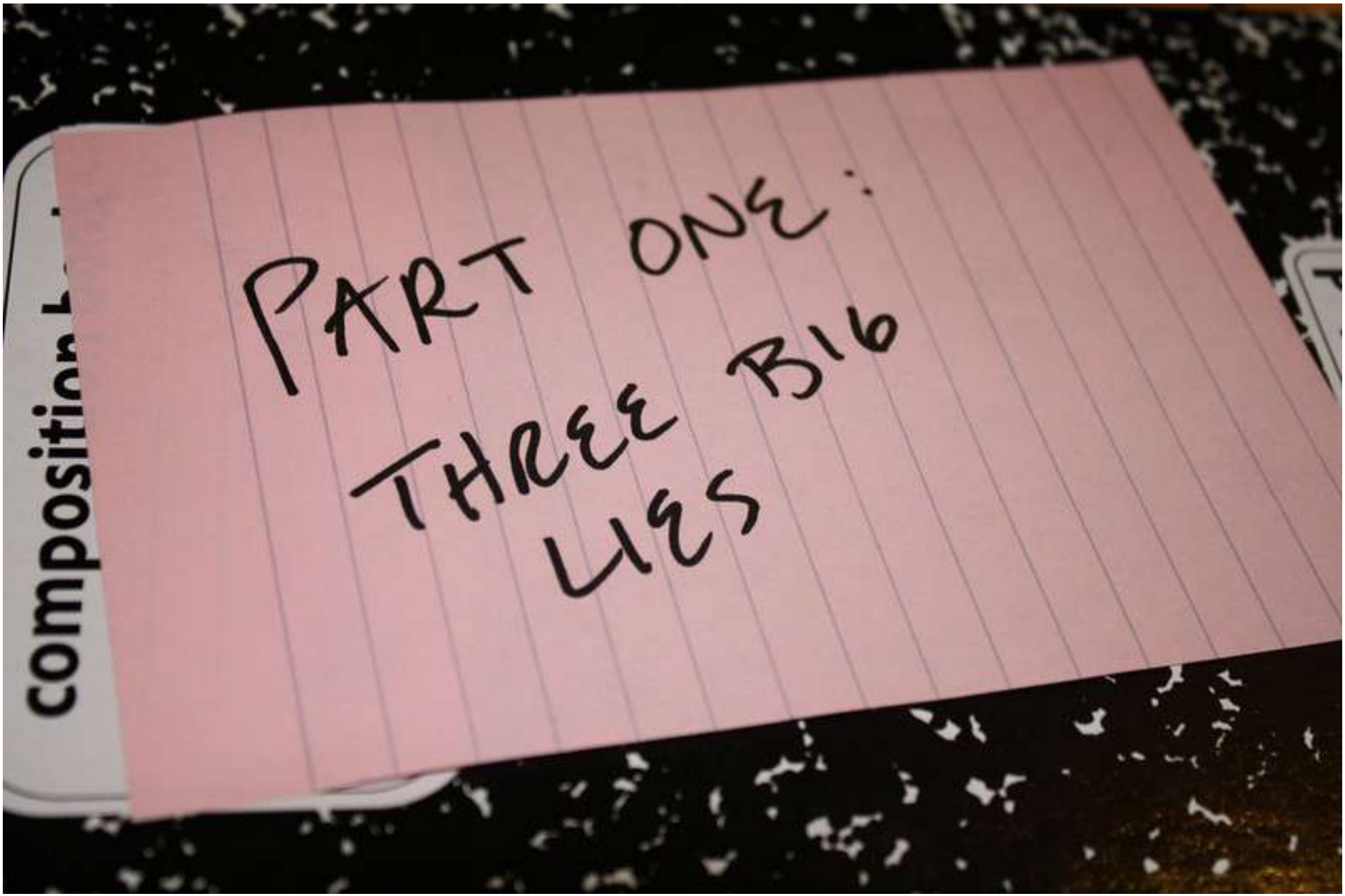

THREE BIG LIES
(Typical design failures in game programming)
Mike Acton
macton@ insomniac games.com
Twitter: @mike_acton

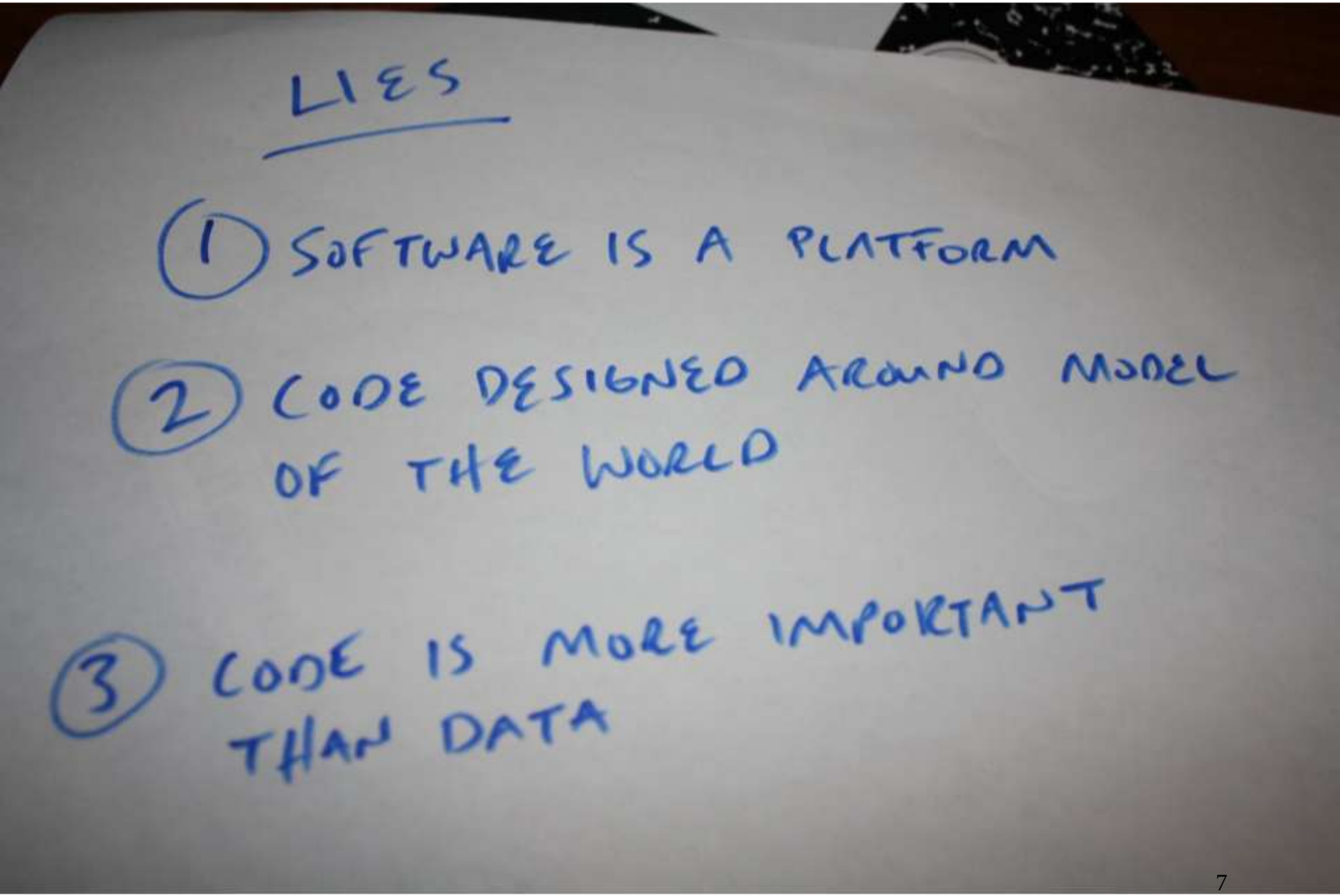

GOAL TODAY:
CHALLENGE TRADITIONAL APPROACHES
SUGGEST ALTERNATIVE DIRECTION.


GOAL TODAY:
CHALLENGE TRADITIONAL APPROACHES
SUGGEST ALTERN
DIRECTION.
"TOO PREACHY"

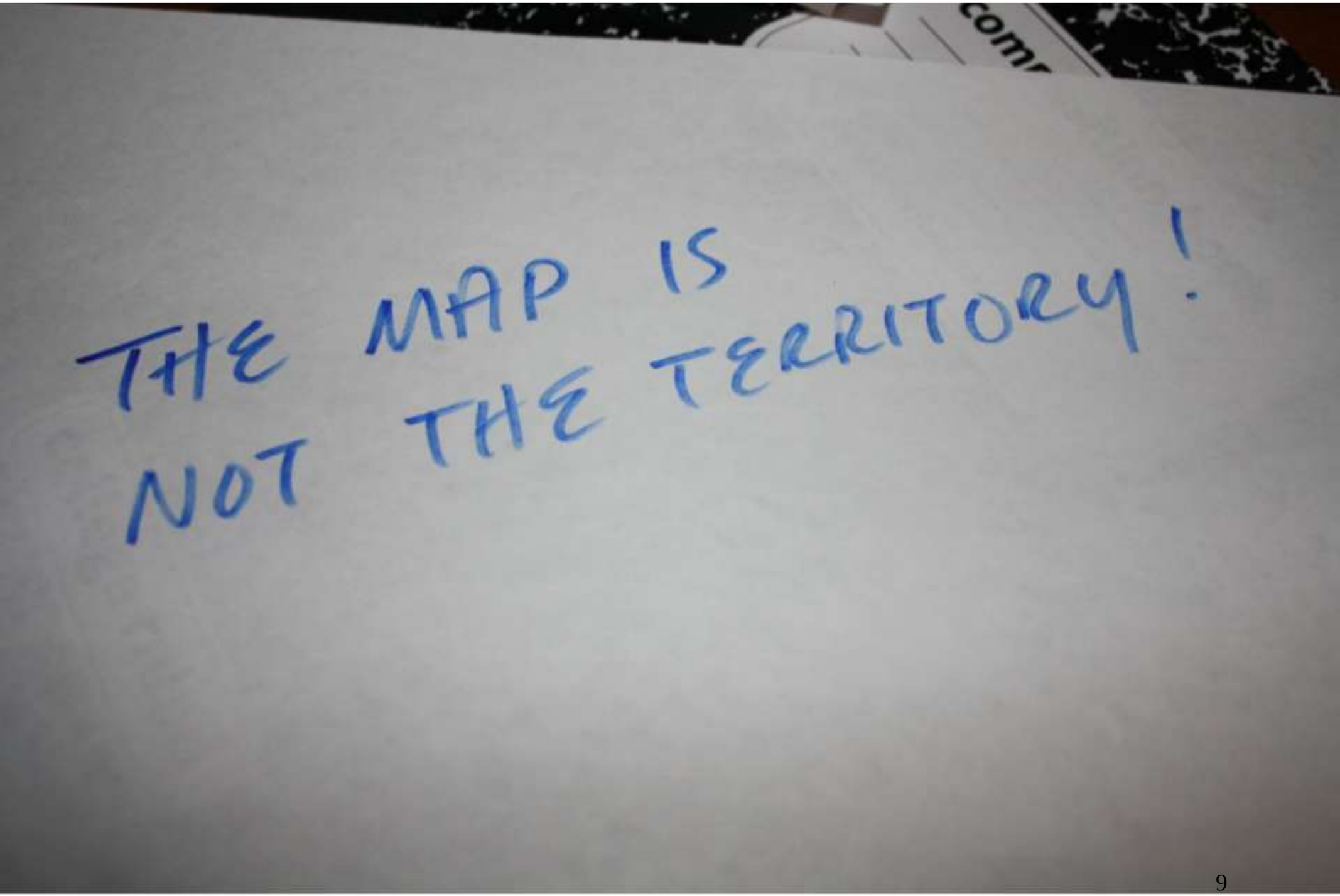

GOAL TODAY:
CHALLENGE TRADITIONAL
APPROACHES
SUGGEST ALT
DIRECTIO
"I WANT TO
SEE MORE
RATCHET CODE
EXAMPLES"

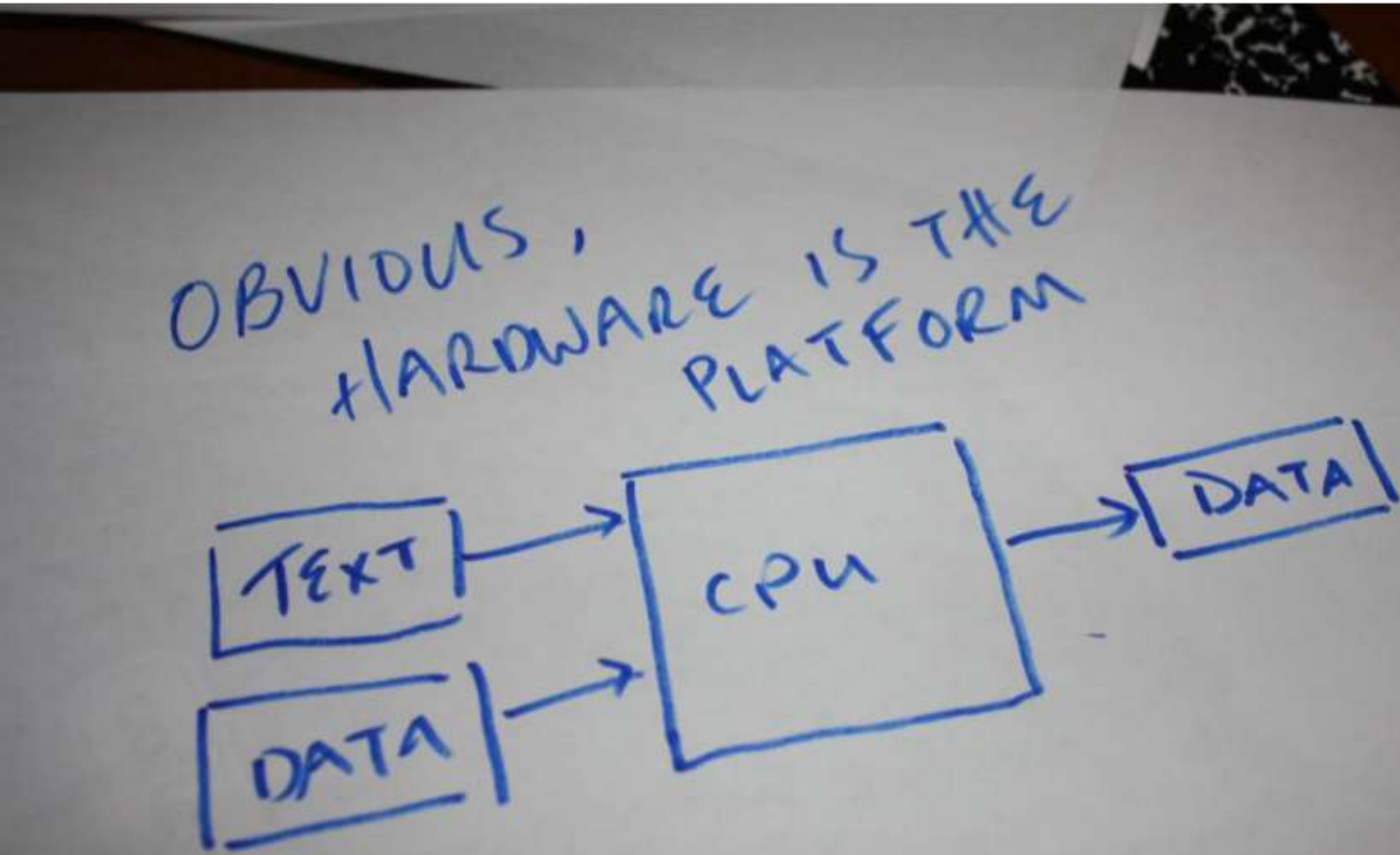

CHALLENGE TRADITIONAL
APPROACH
SUGGEST
"TOO
101"
TO

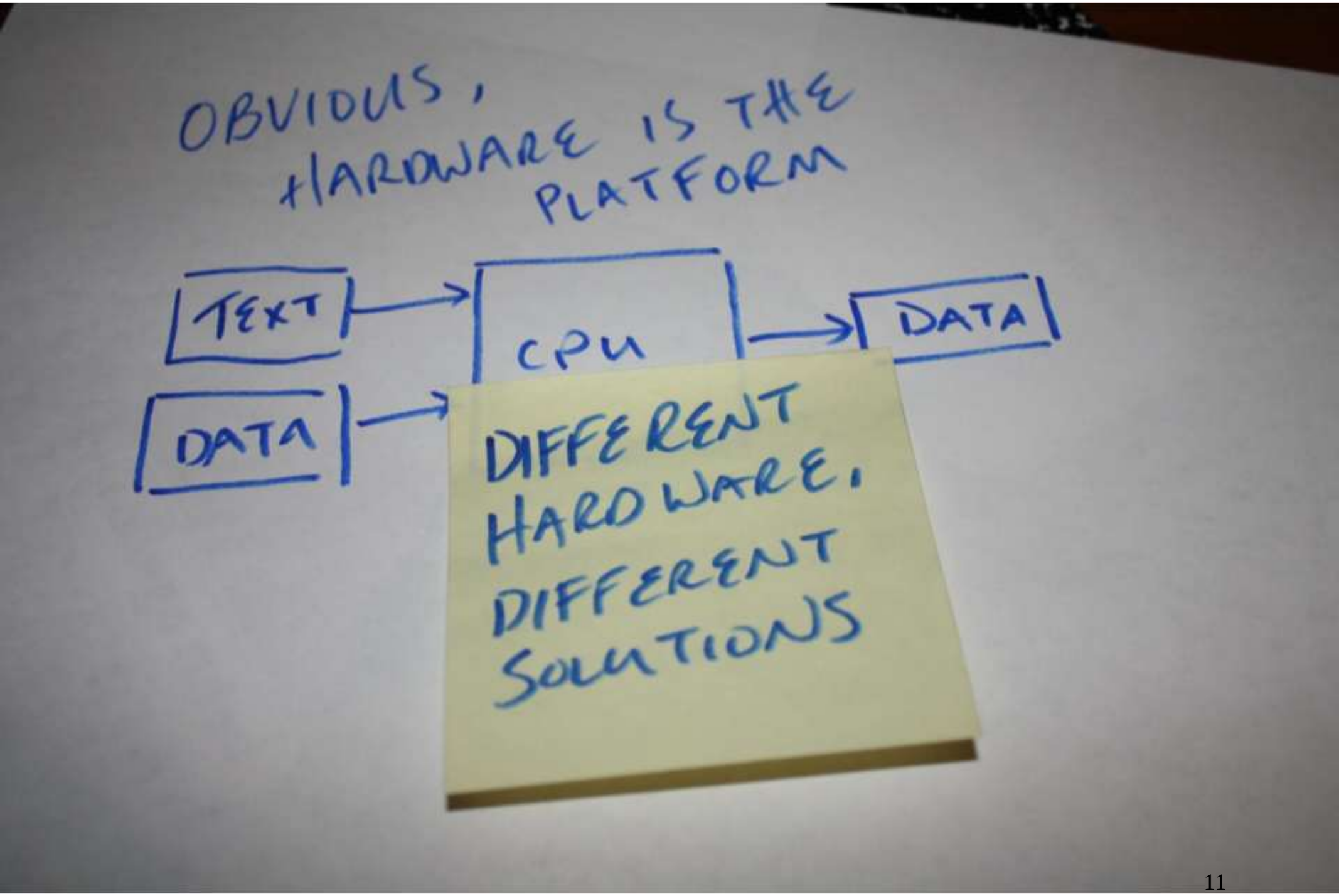

PART ONE:
THREE BIG
LIES


LIES
1 SOFTWARE IS A PLATFORM
2 CODE DESIGNED AROUND MODEL OF THE WORLD
3 CODE IS MORE IMPORTANT THAN DATA

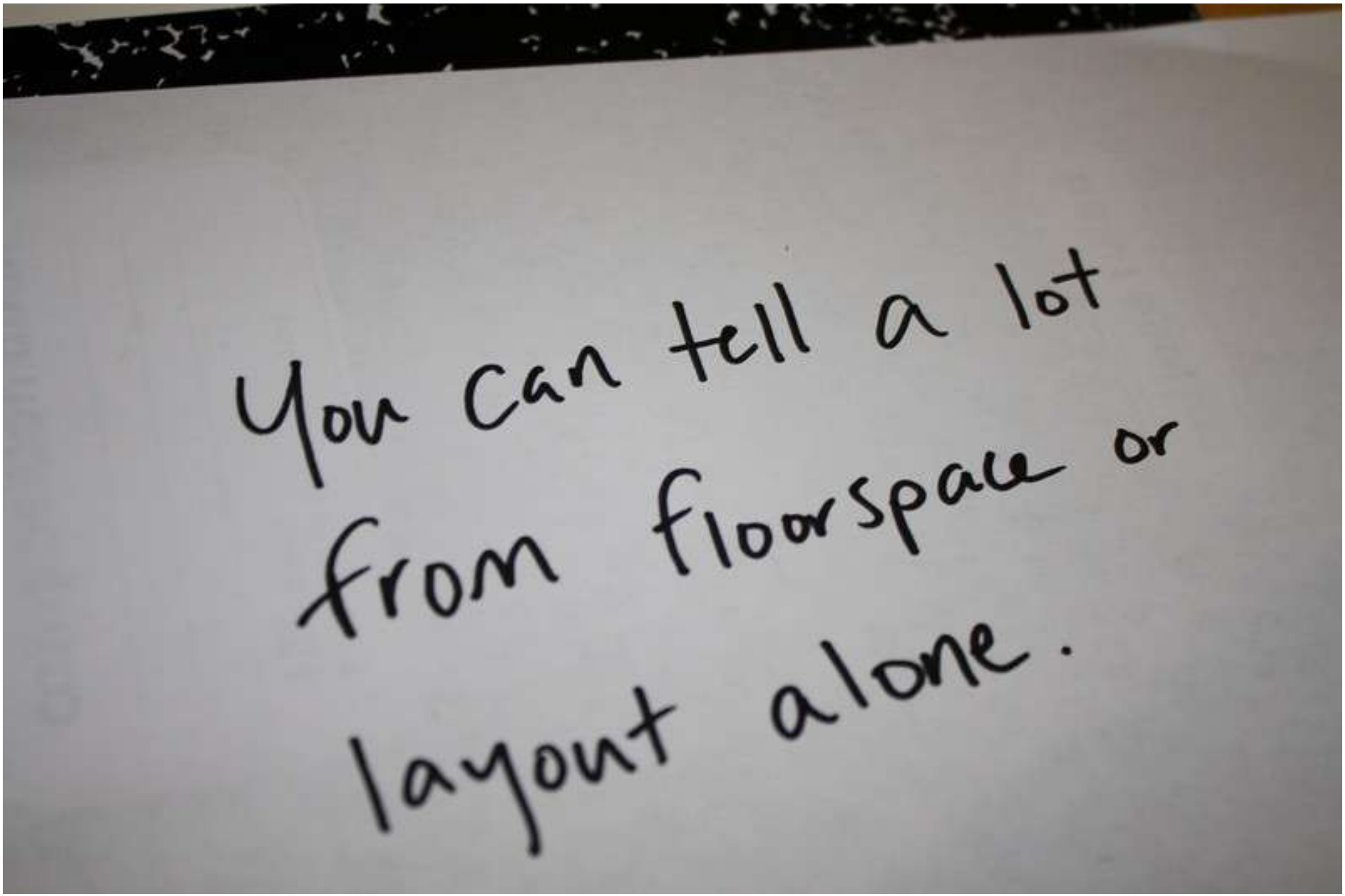

LIE #1:
SOFTWARE IS
A PLATFORM

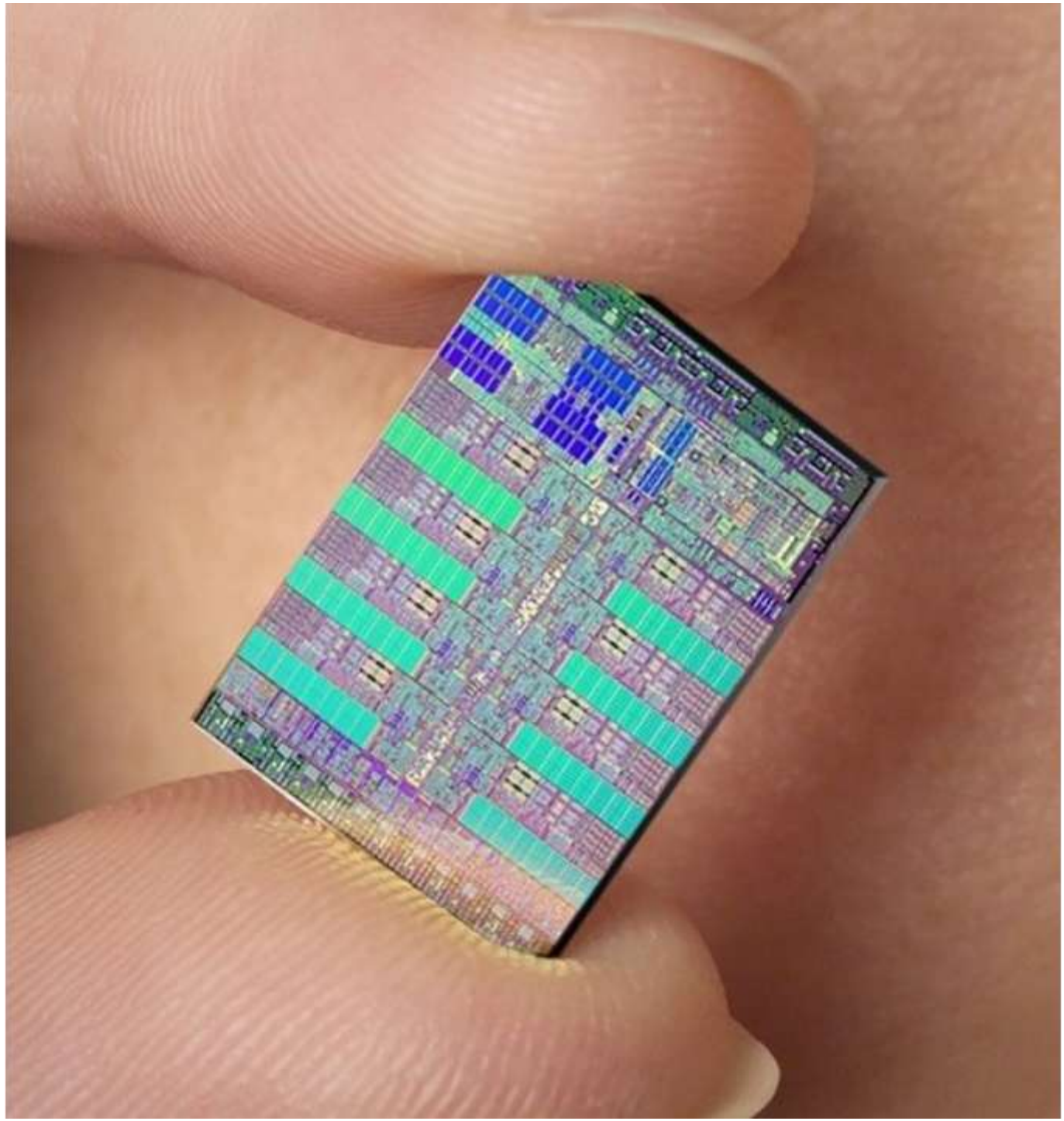

THE MAP IS
NOT THE TERRITORY!


OBVIOUS,
HARDWARE IS THE PLATFORM
TEXT
DATA
CPU
DATA

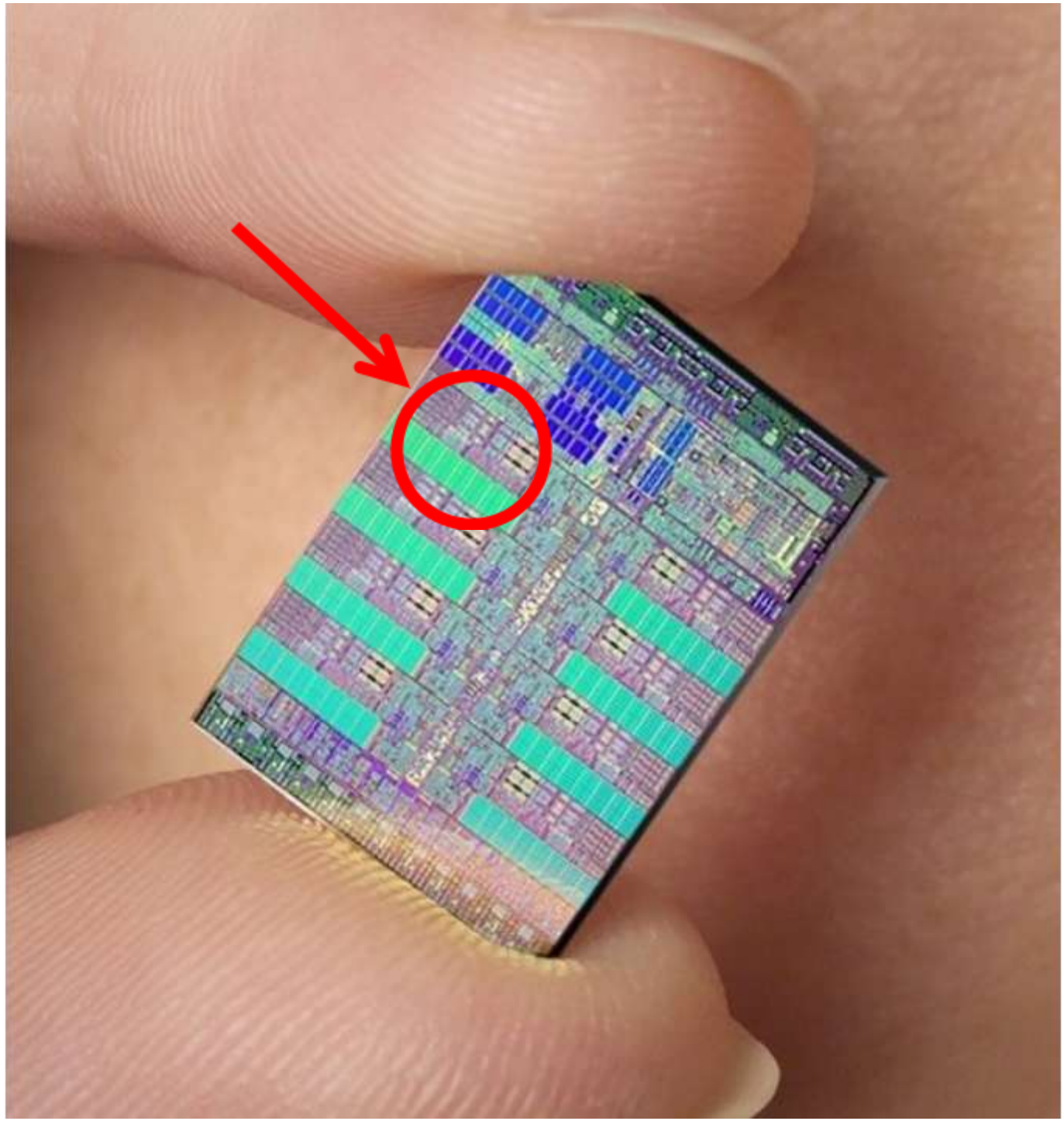

OBVIOUS,
HARDWARE IS THE PLATFORM
TEXT
DATA
CPU
DATA
DIFFERENT HARDWARE, DIFFERENT SOLUTIONS


OBVIOUS,
HARDWARE IS THE PLATFORM
TEXT
CPU
DATA
DATA
DIFFERENT
6502
x86, ARM
CELL, PPC
ATI 5870

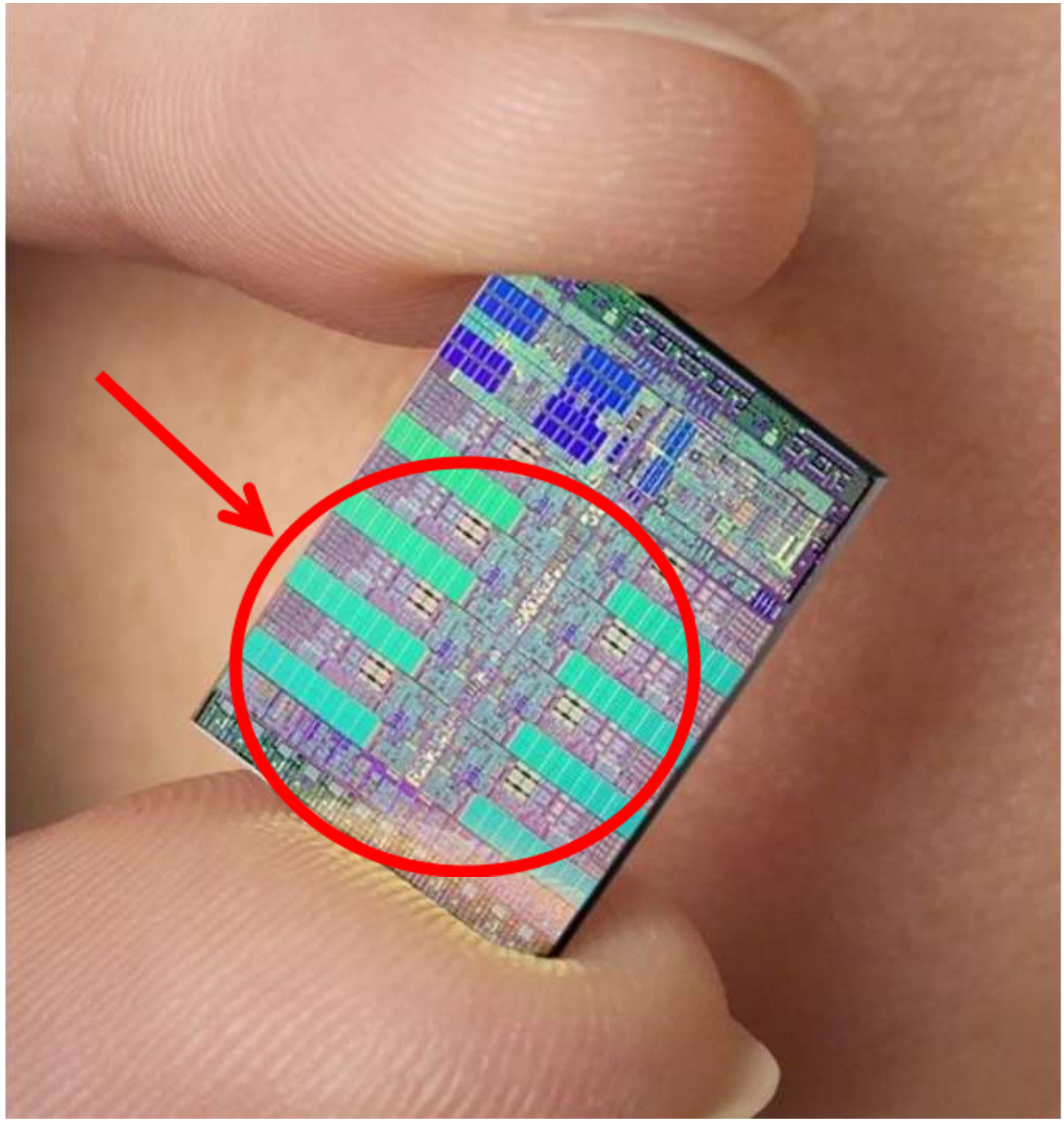

You can tell a lot
from floorspace or
layout alone.



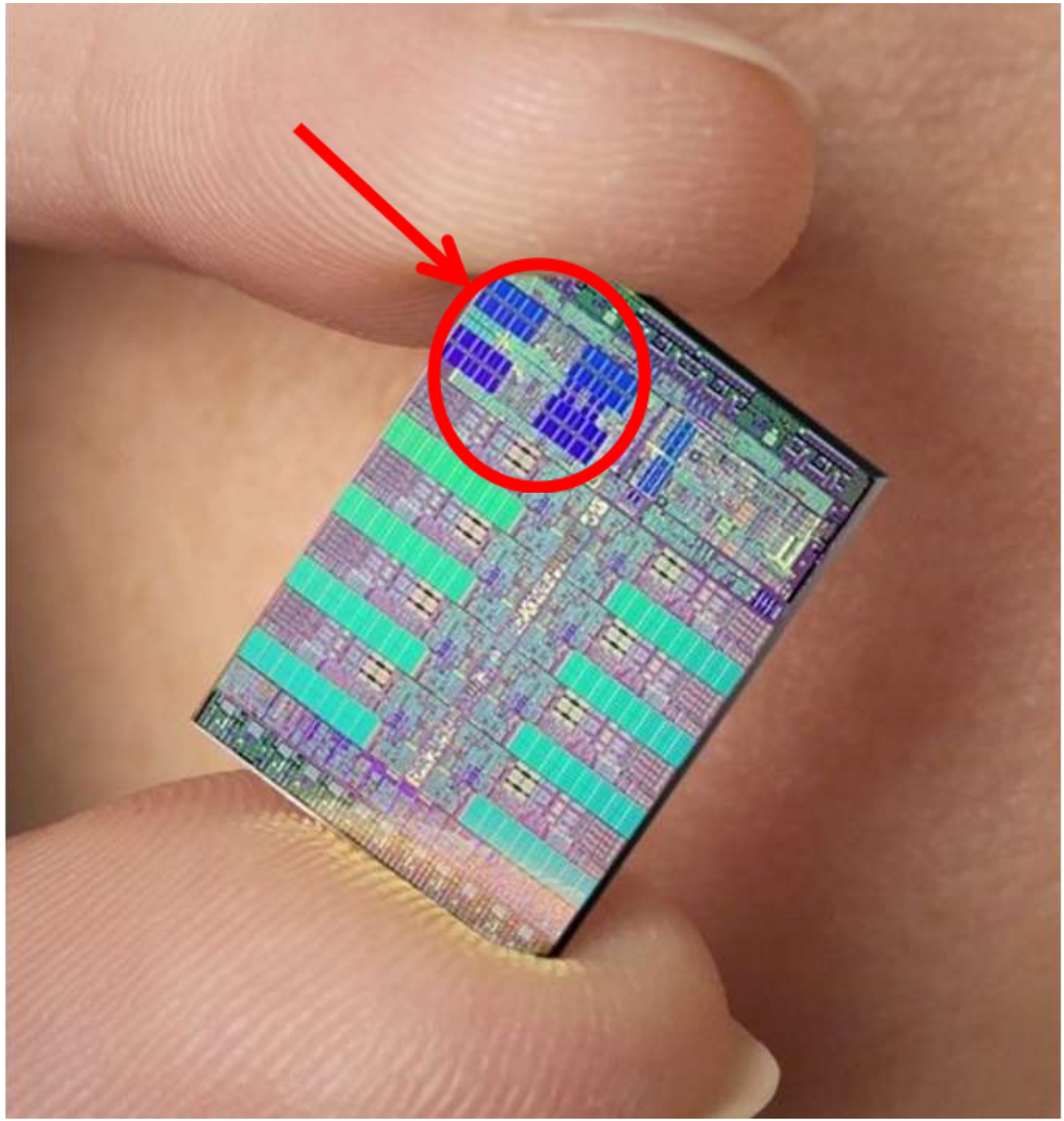
Cell Processor



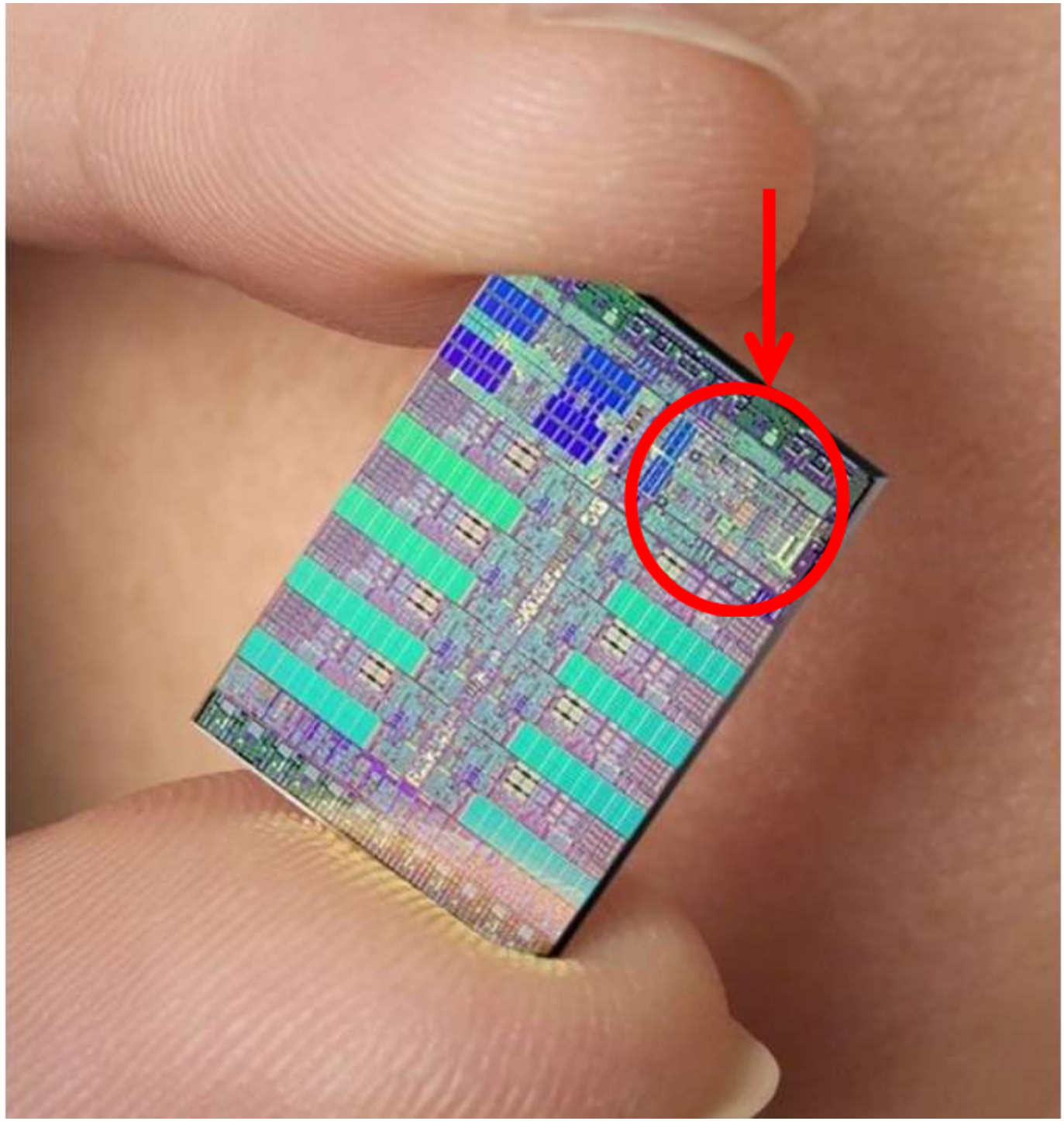
Cell Processor



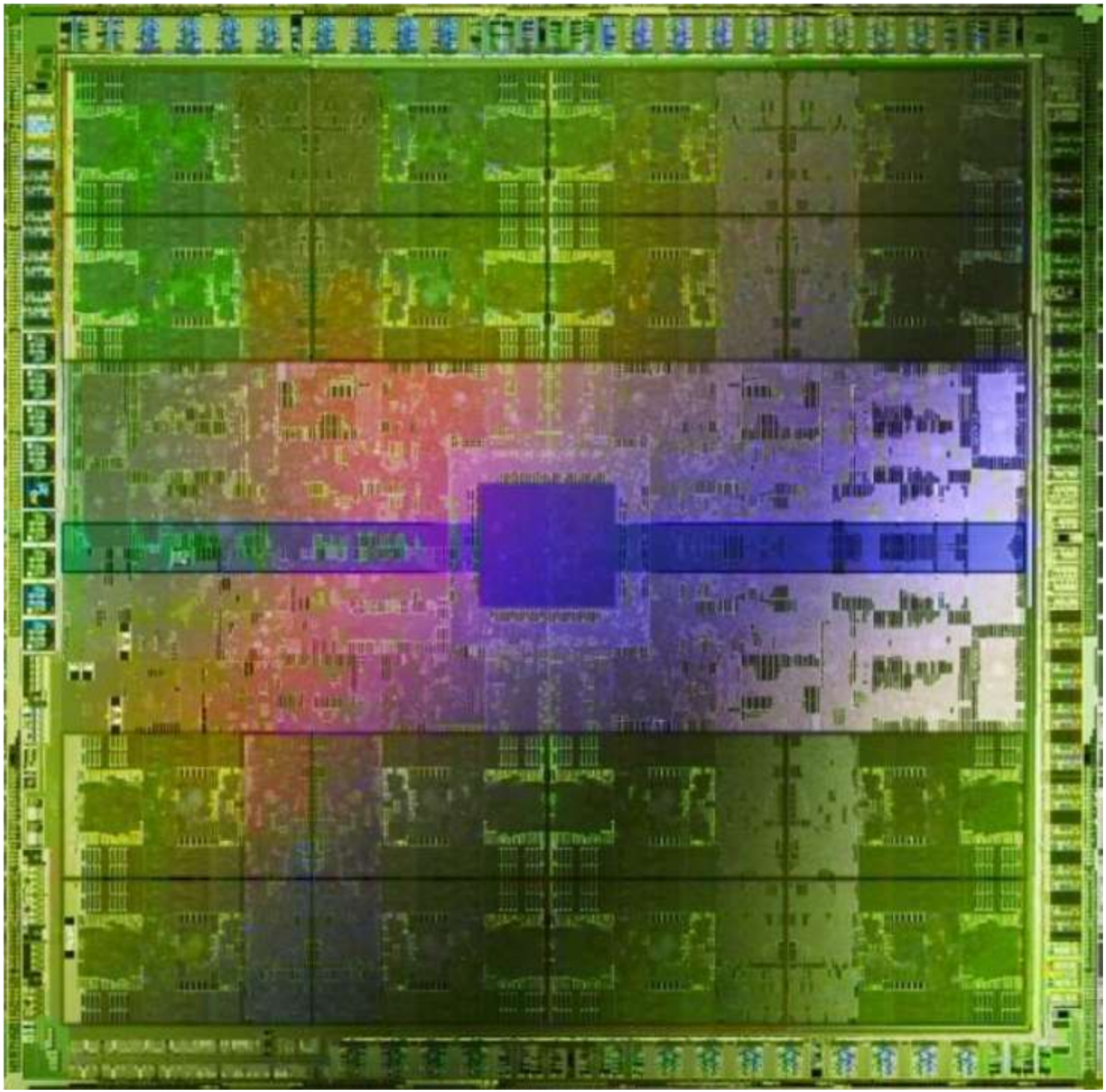
Cell Processor



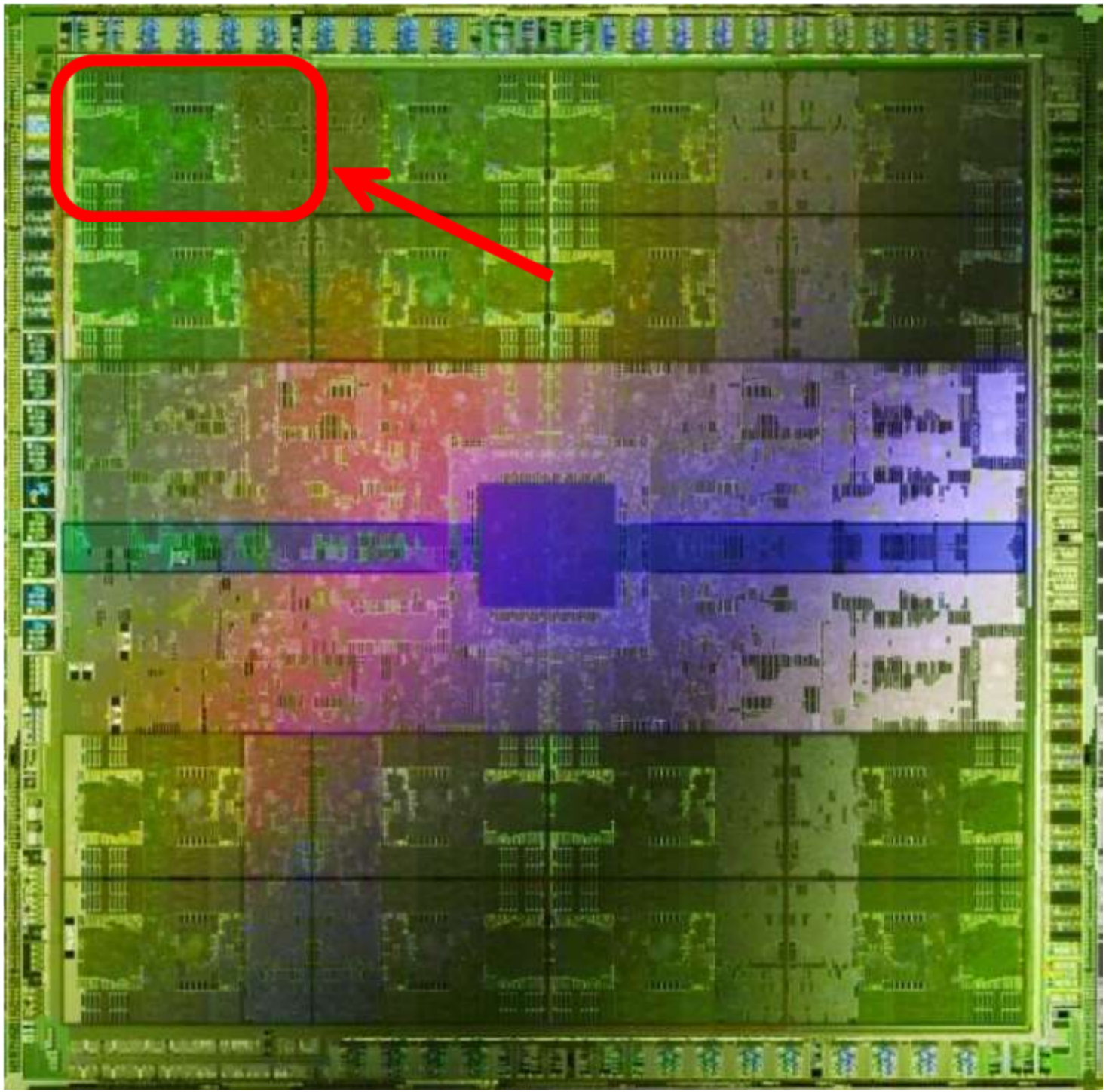
Cell Processor



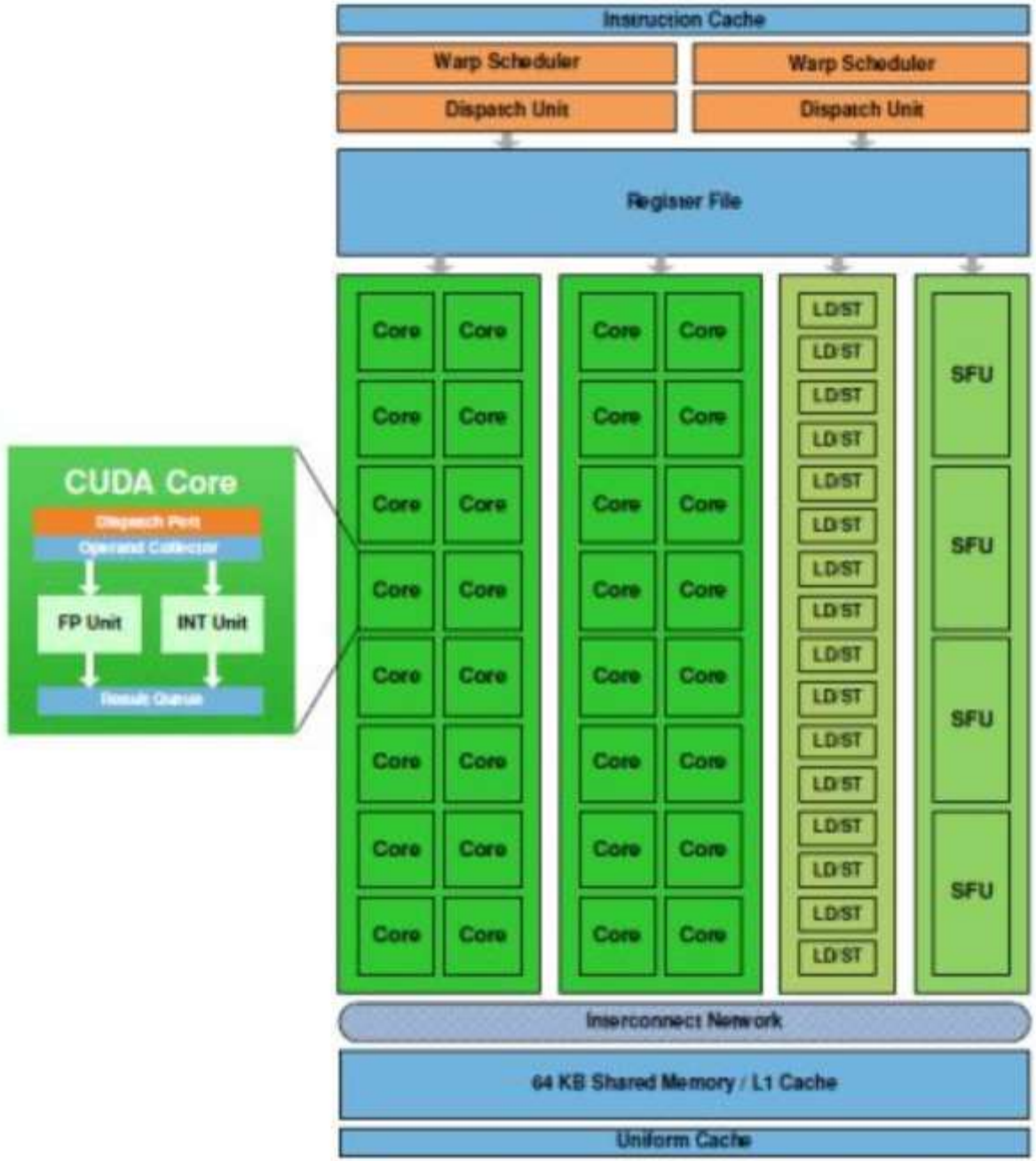
Cell Processor



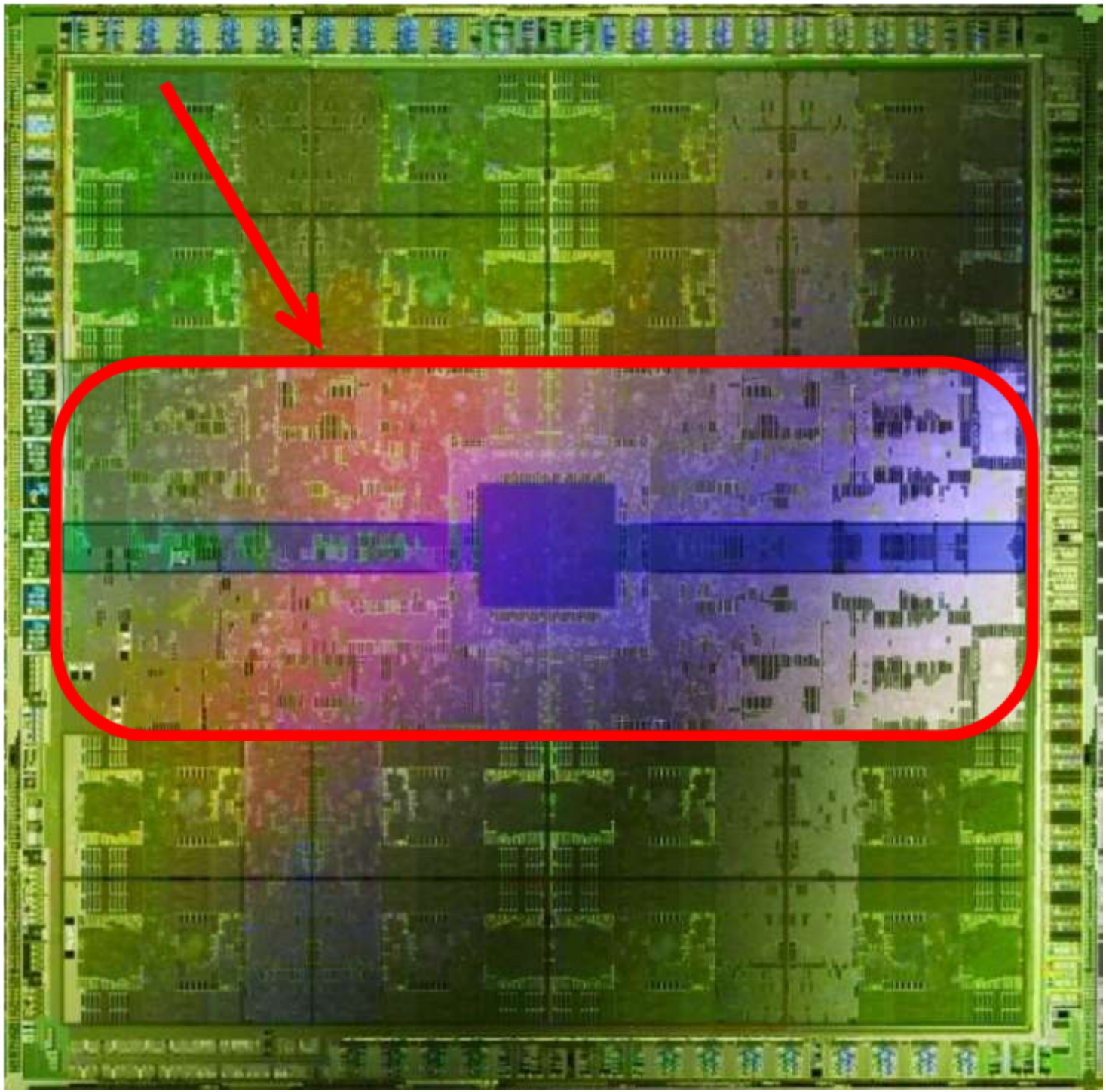
NVidia Fermi (GF100)



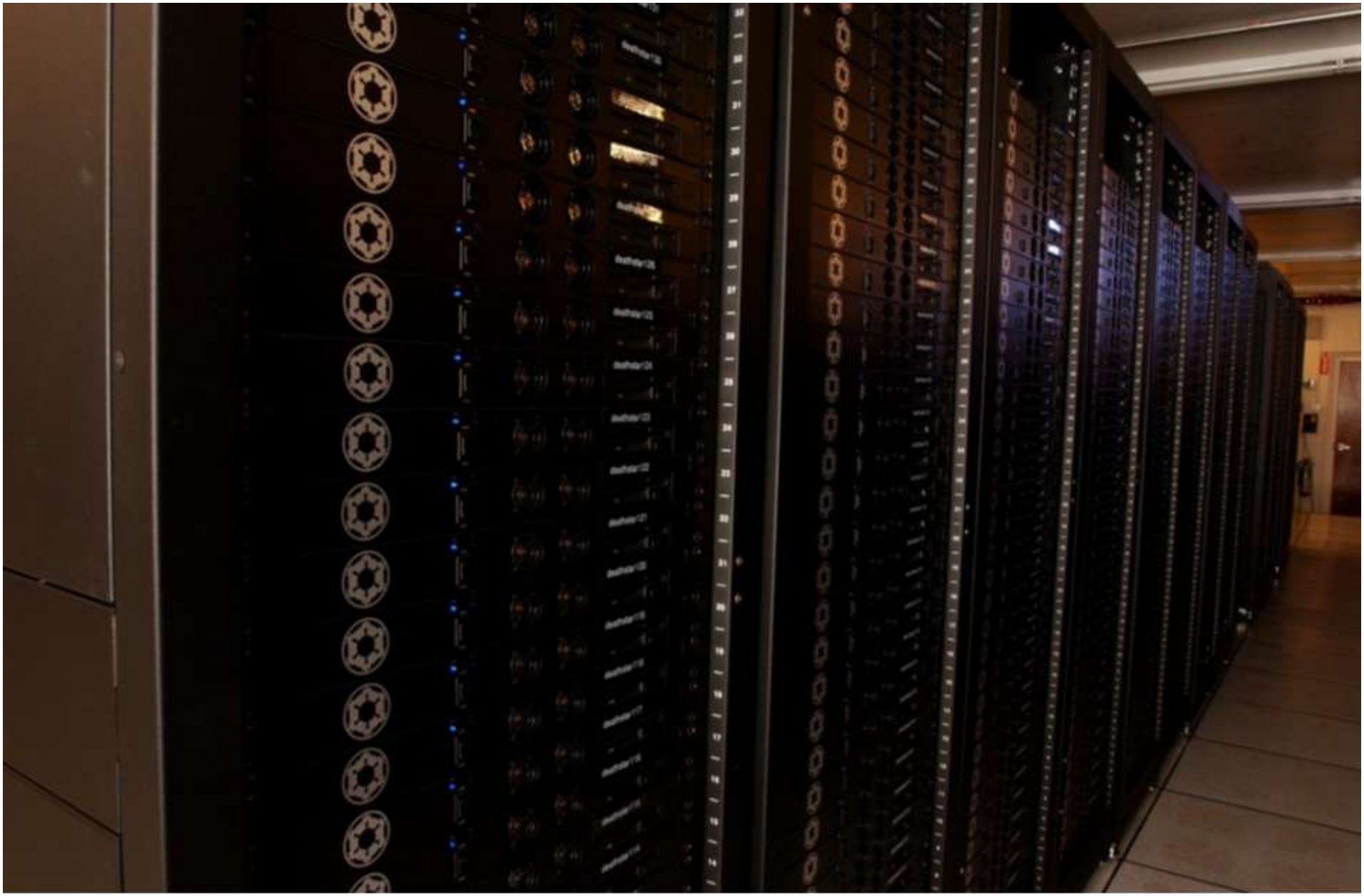
NVidia Fermi (GF100)



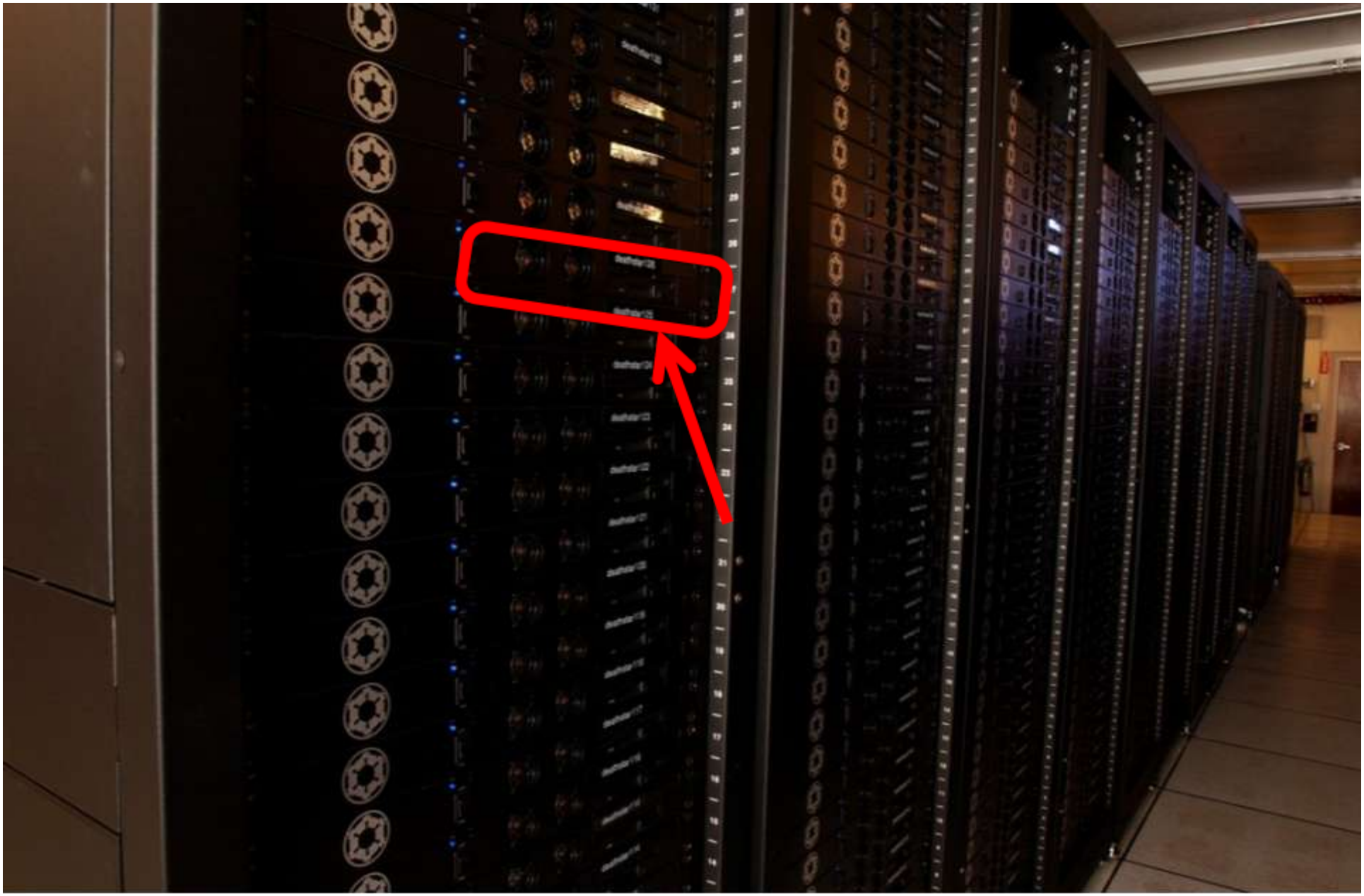
Fermi Streaming Multiprocessor (SM)



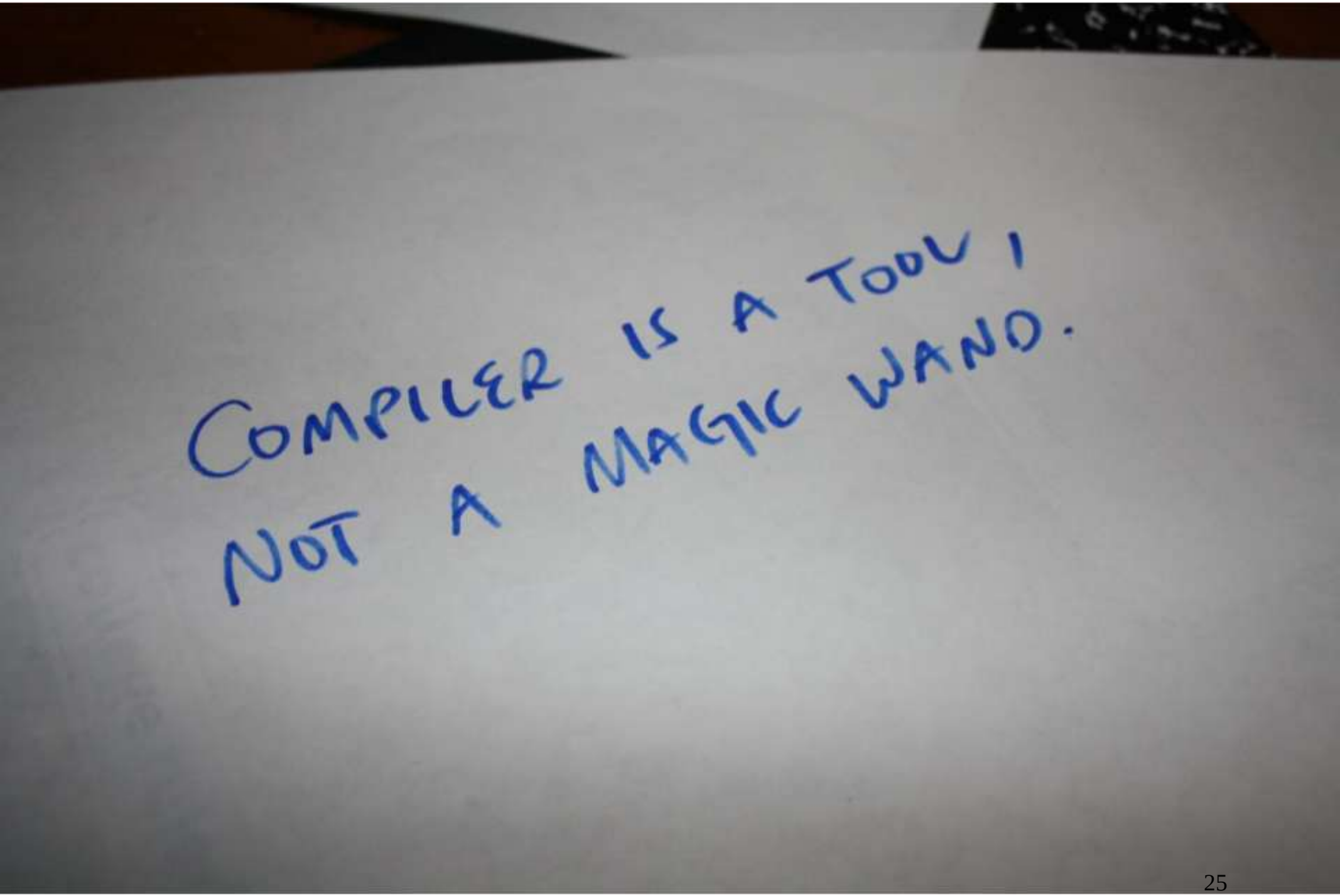
NVidia Fermi (GF100)

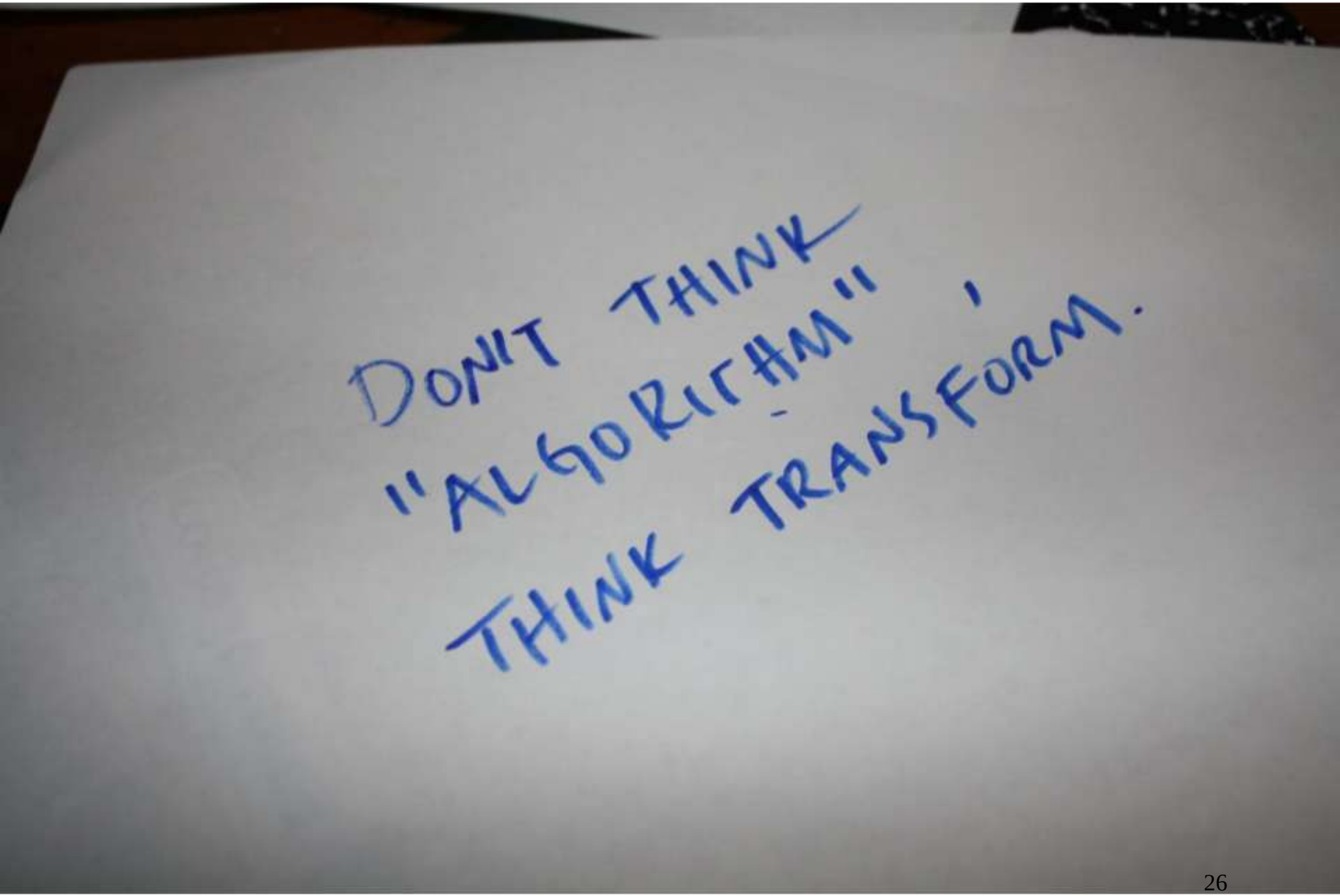


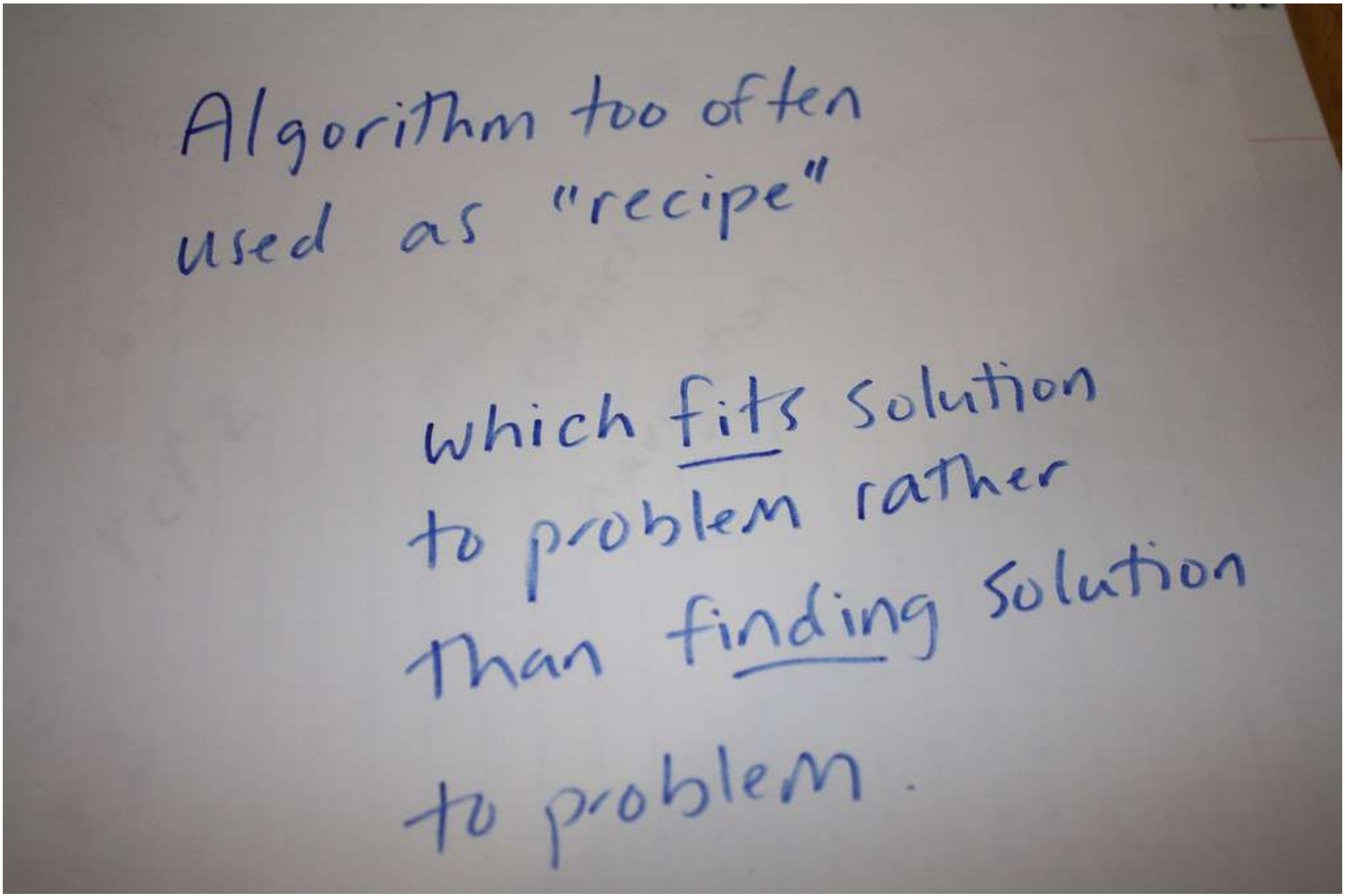
ILM's RackSaver Linux renderfarm
(1,500 processors, circa 2003)



ILM's RackSaver Linux renderfarm
(1,500 processors, circa 2003)

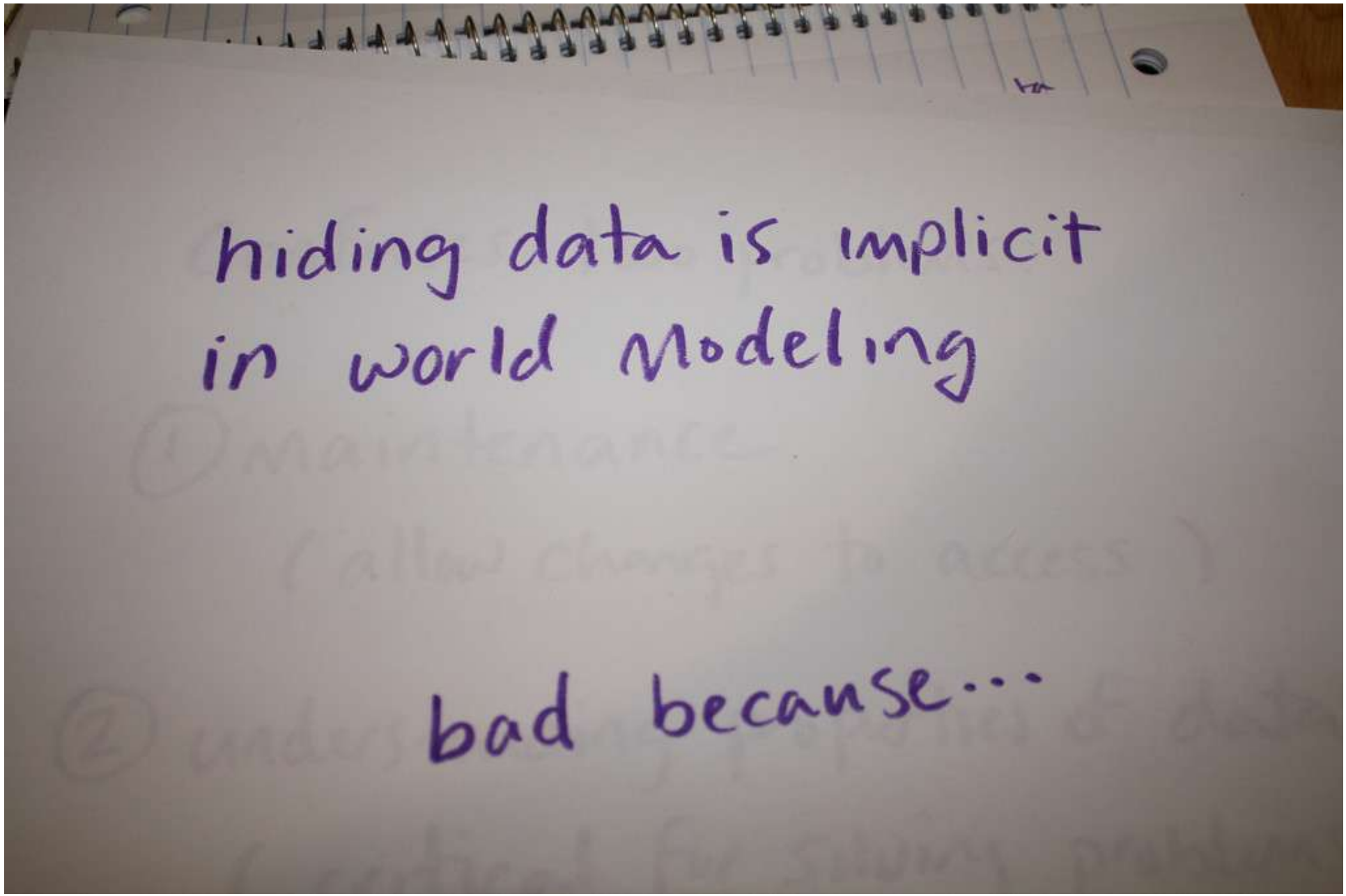

COMPILER IS A TOOL !
NOT A MAGIC WAND.

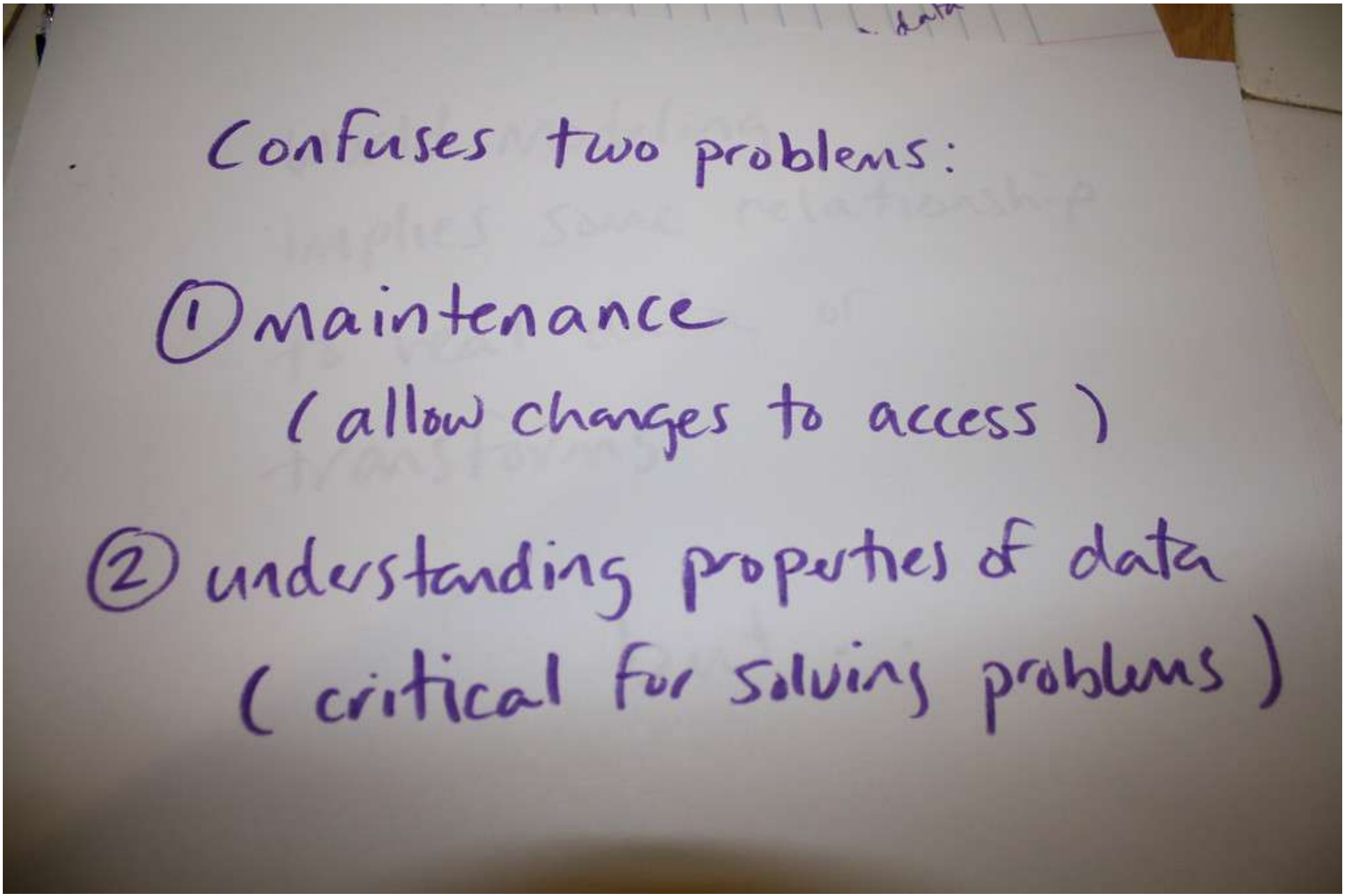

DON'T THINK
"ALGORITHM"
THINK TRANSFORM.

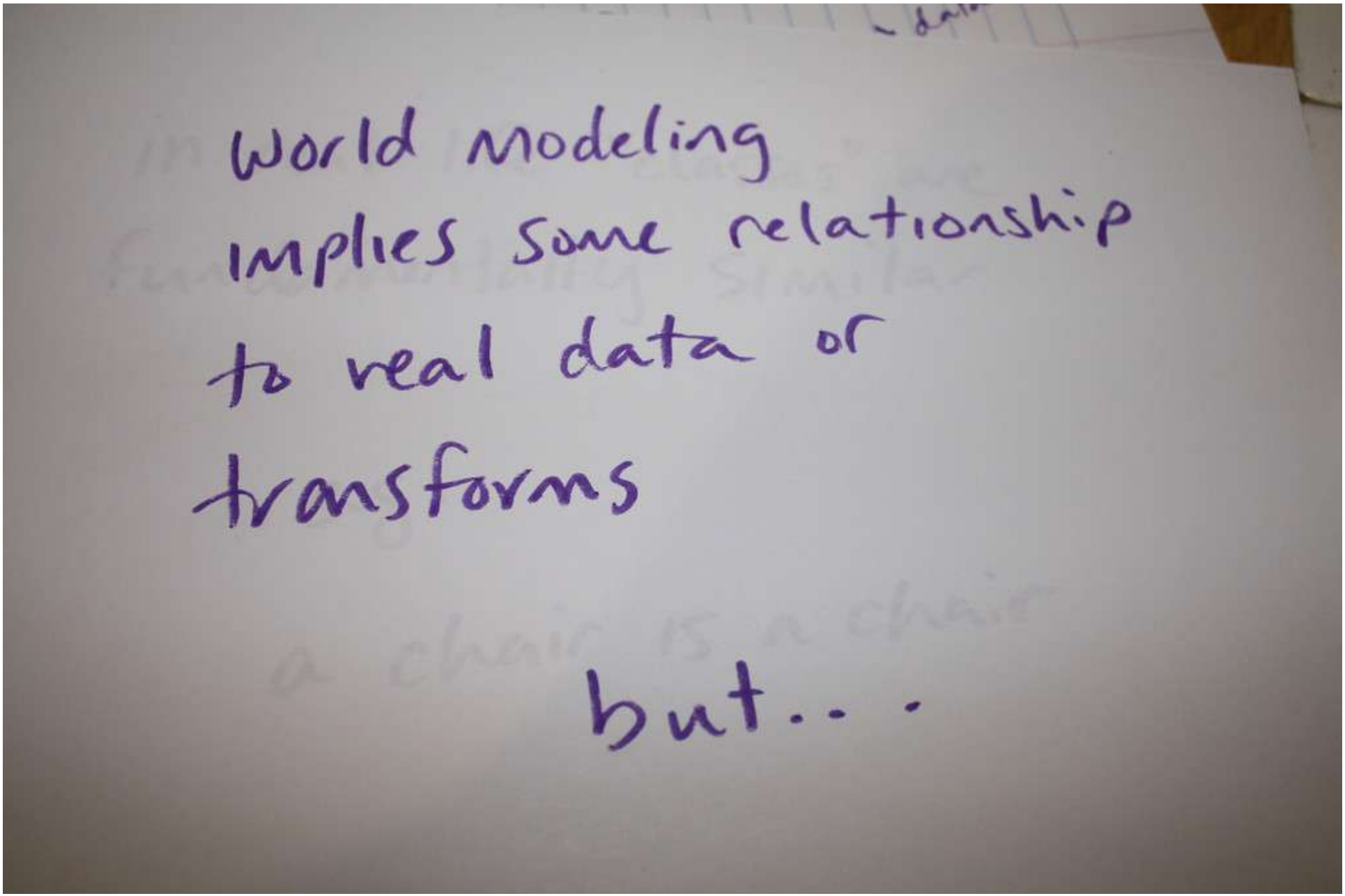

Algorithm too often
used as "recipe"
which fits solution
to problem rather
than finding solution
to problem.

LIE #2:

CODE DESIGNED AROUND MODEL OF THE WORLD


hiding data is implicit
in world Modeling
bad becanse...

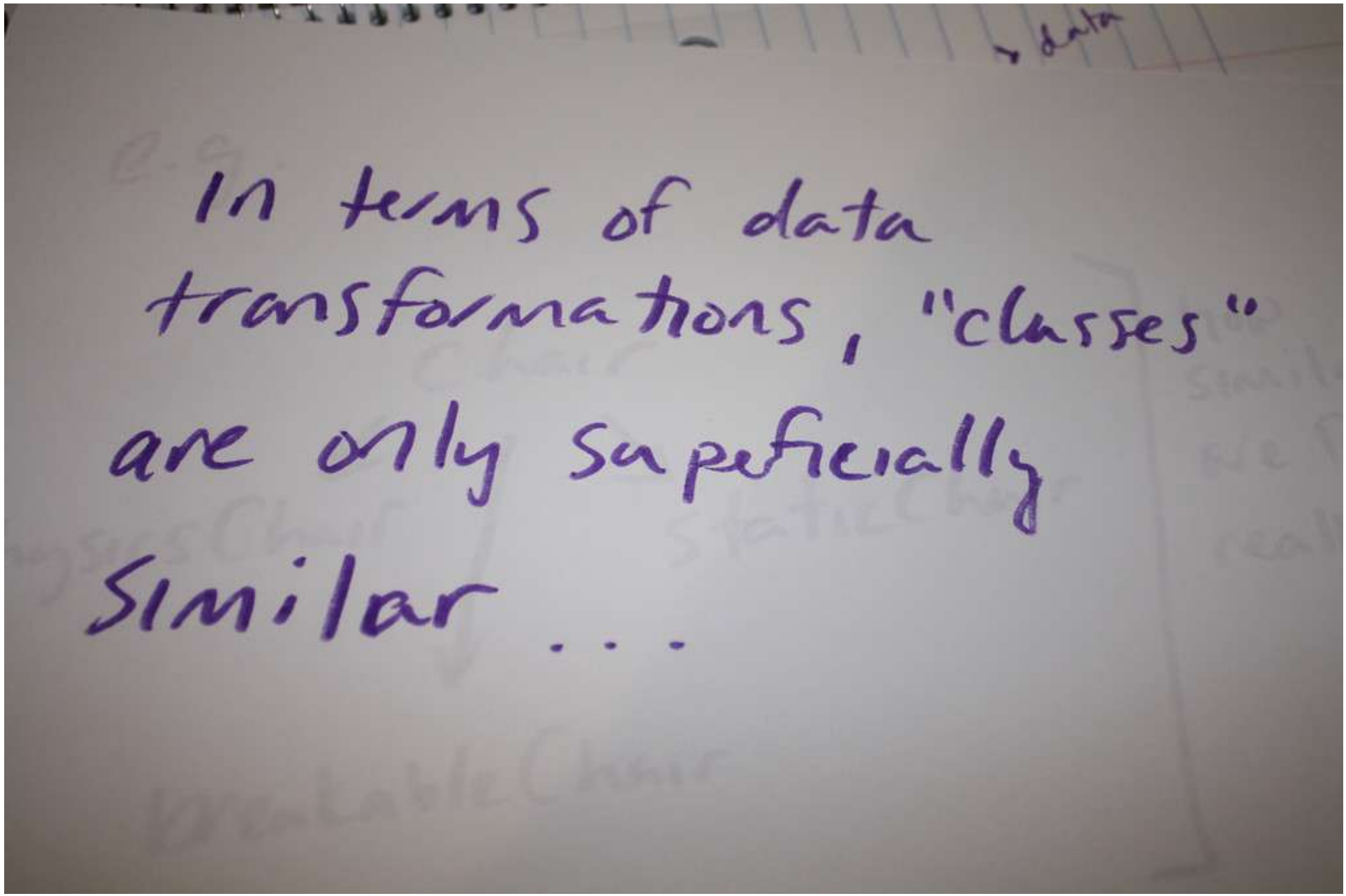

Confuses two problems:
① maintenance
(allow changes to access)
② understanding properties of data
(critical for solving problems)

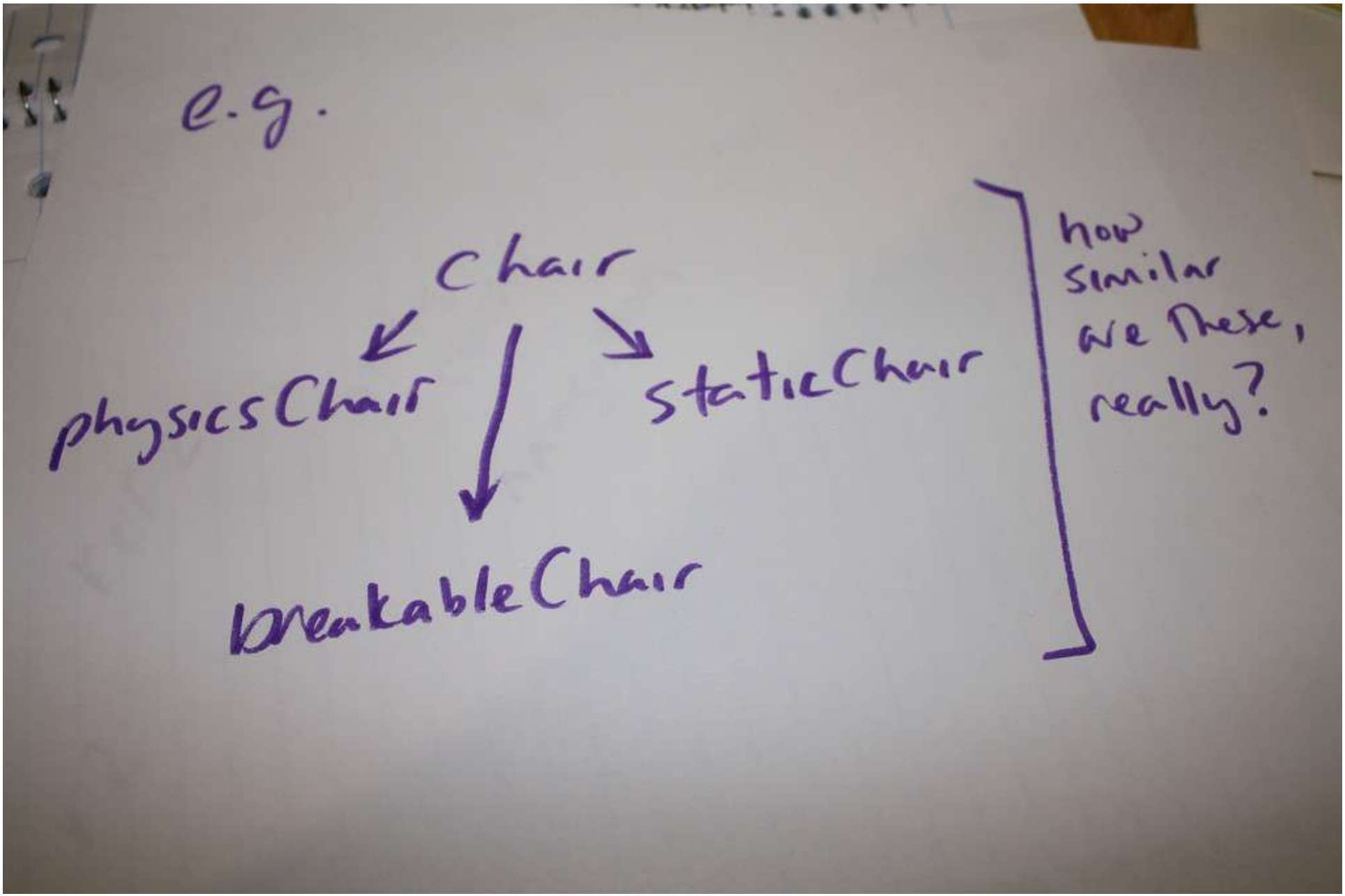

World modeling
implies some relationship
to real data or
transforms
but...

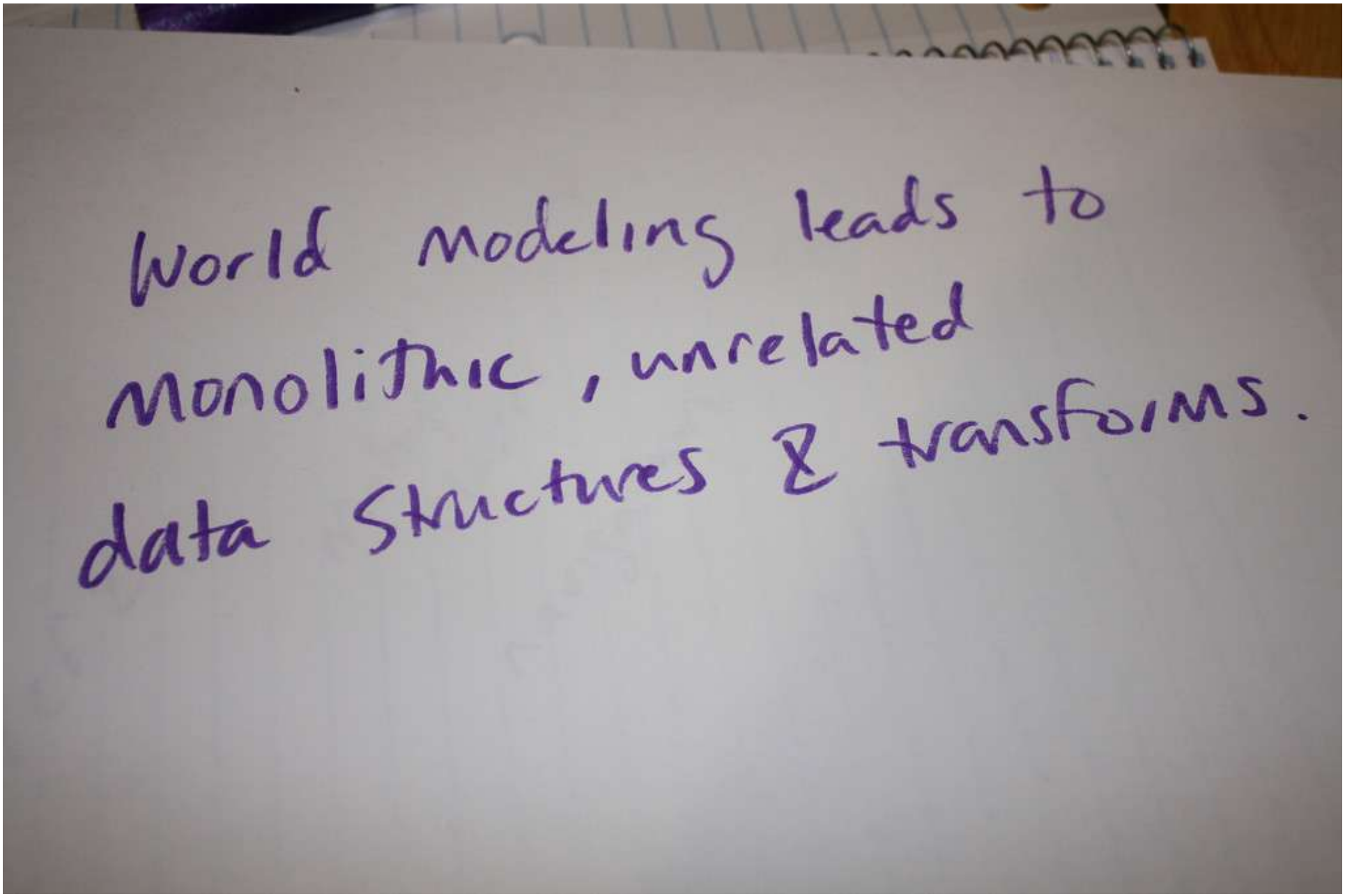
in real life "classes" are fundamentally similar

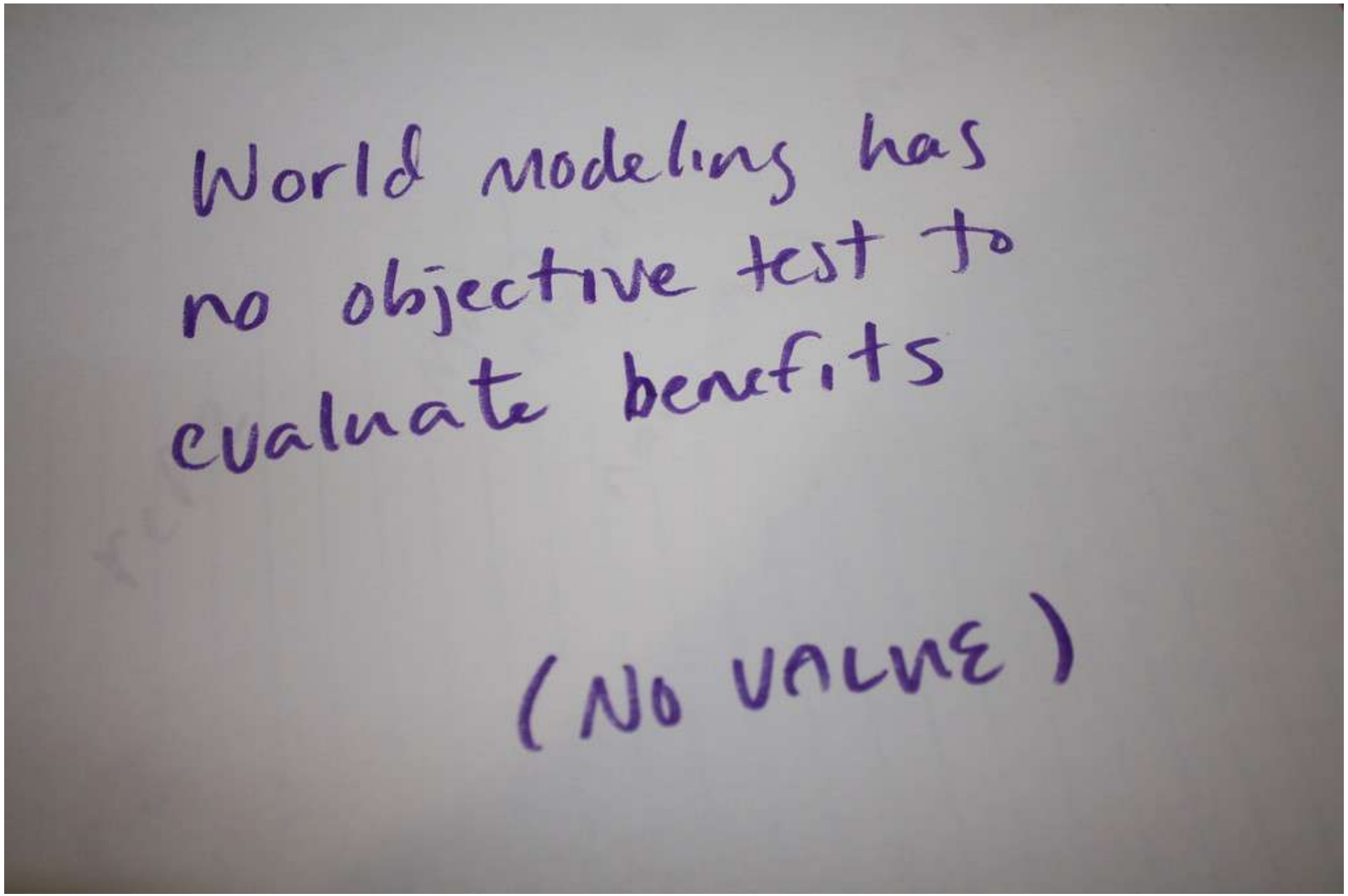
e.g.

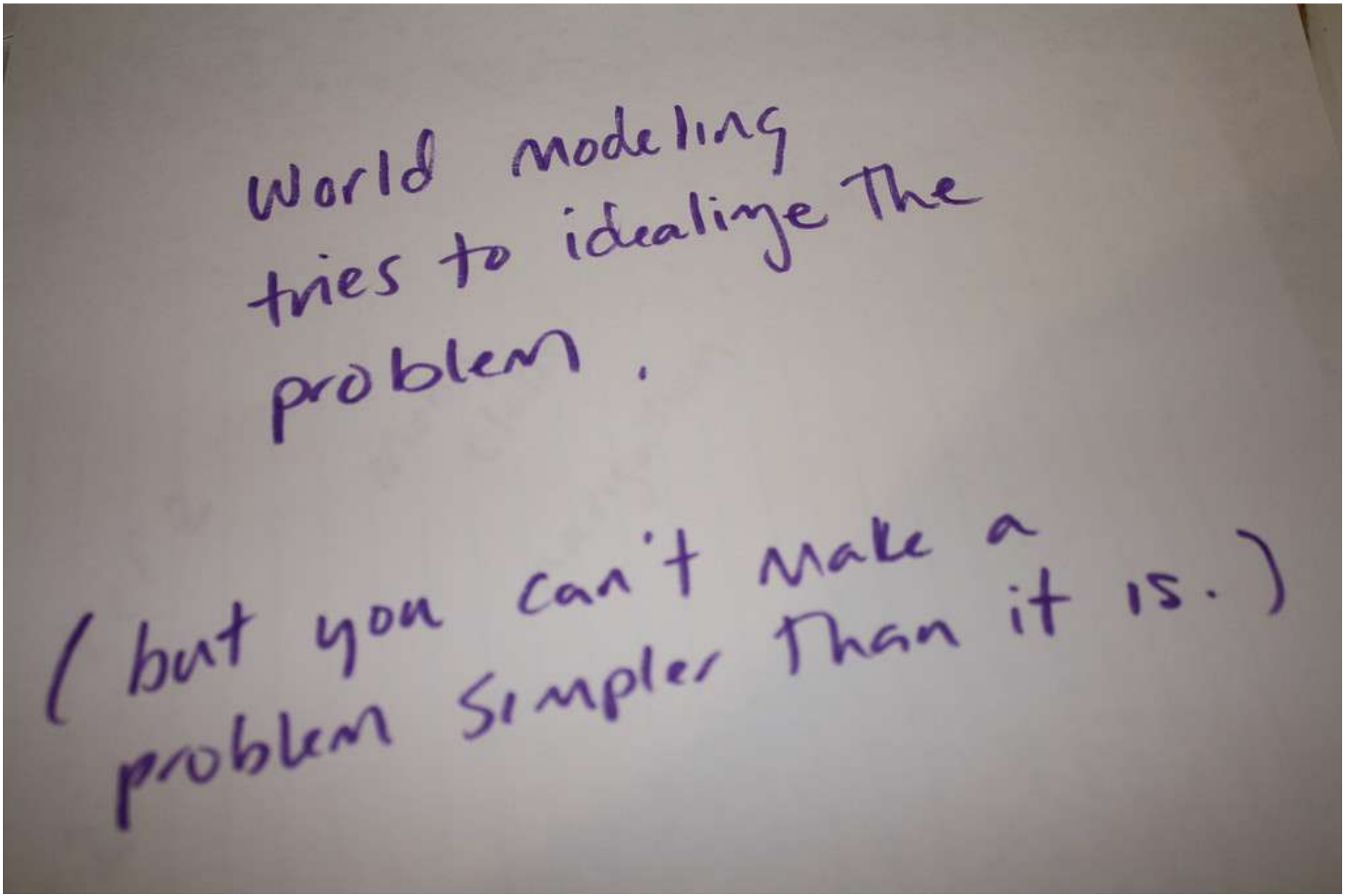
a chair is a chair

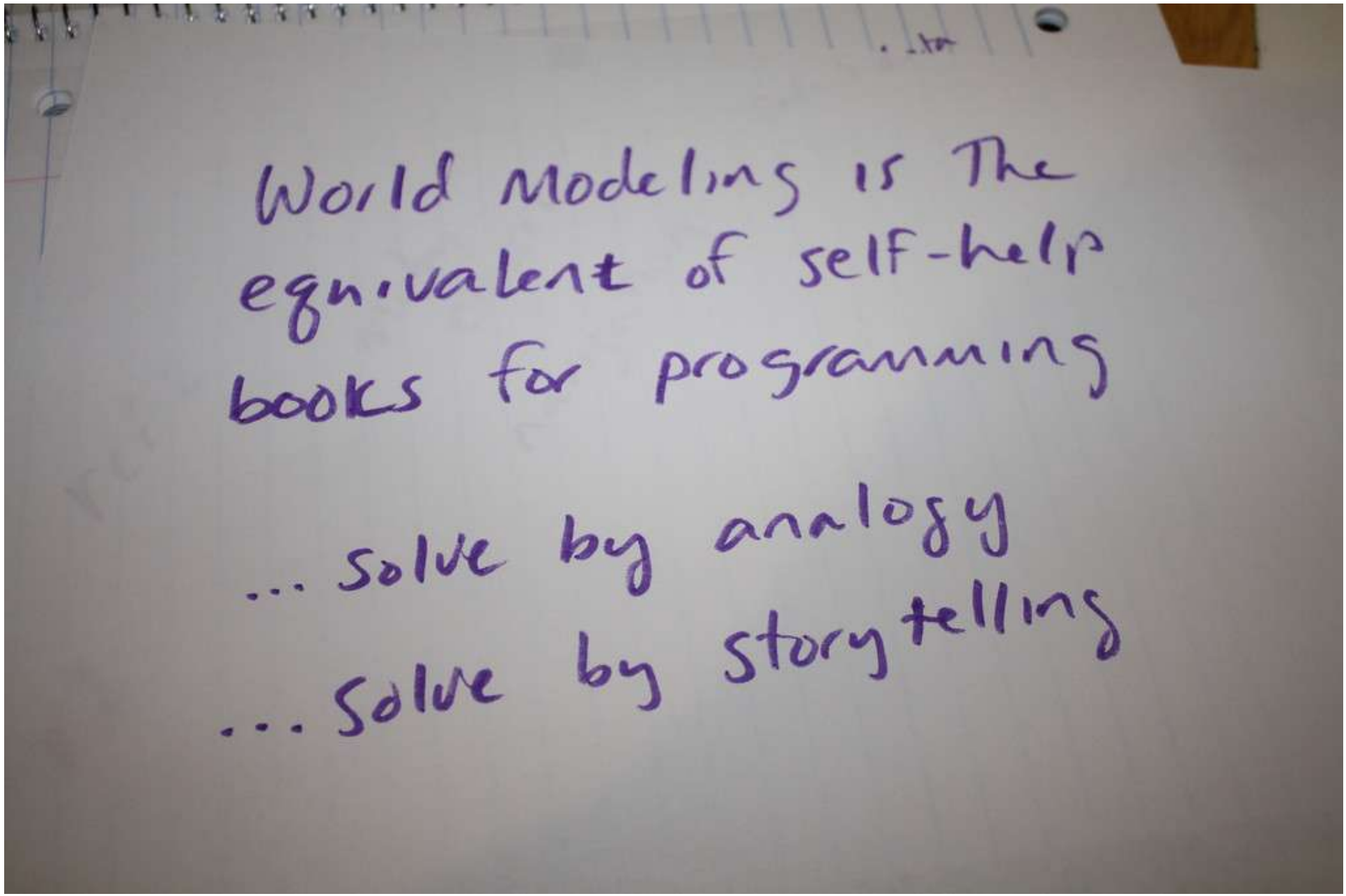

data
In terms of data
transformations, "classes"
are only superficially
similar . . .

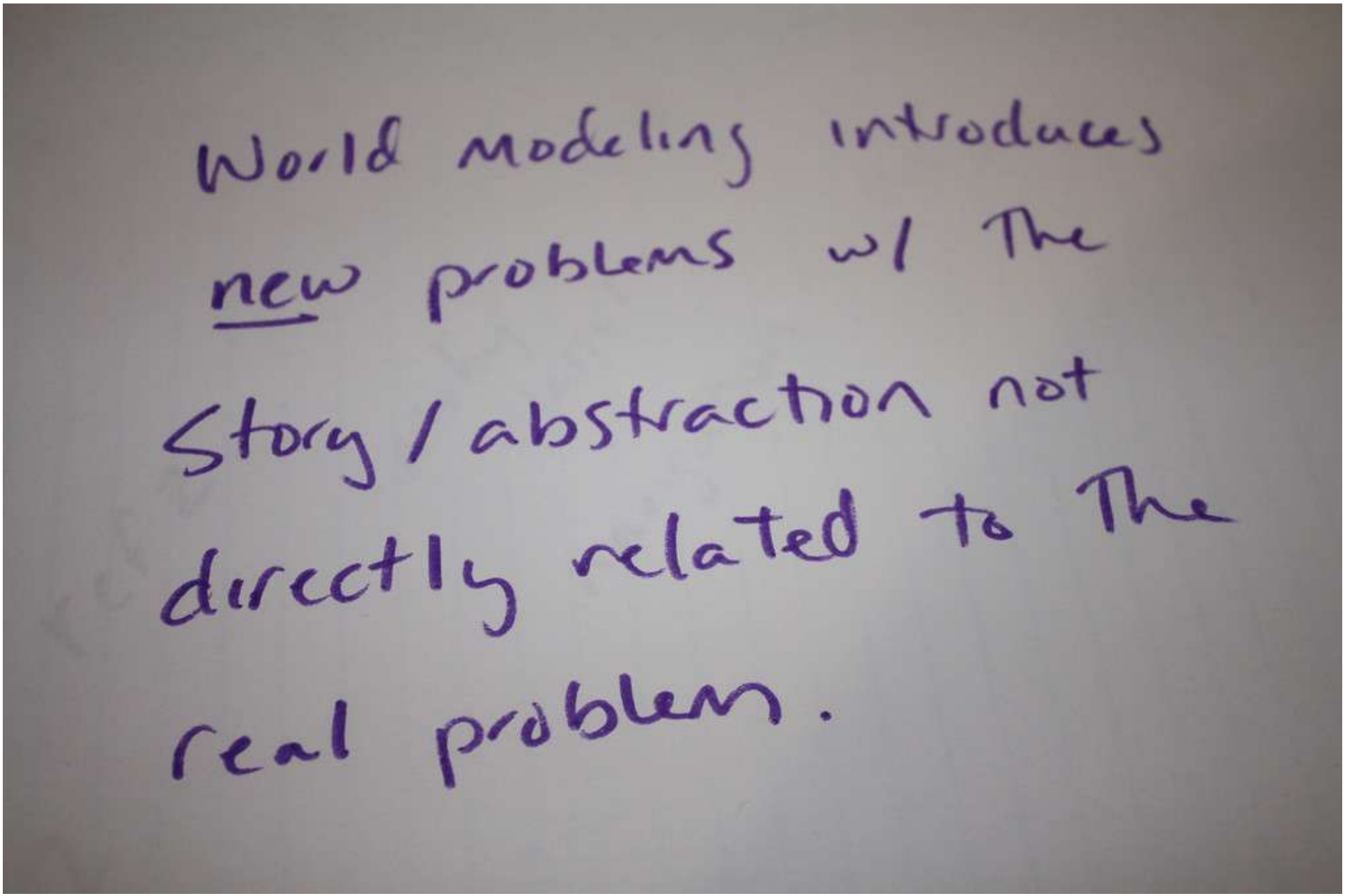

e.g.
Chair
physicsChair
staticChair
breakableChair
how similar are these, really?


World modeling leads to
monolithic, unrelated
data structures & transforms.


World modeling has
no objective test to
evaluate benefits
(NO VALUE)

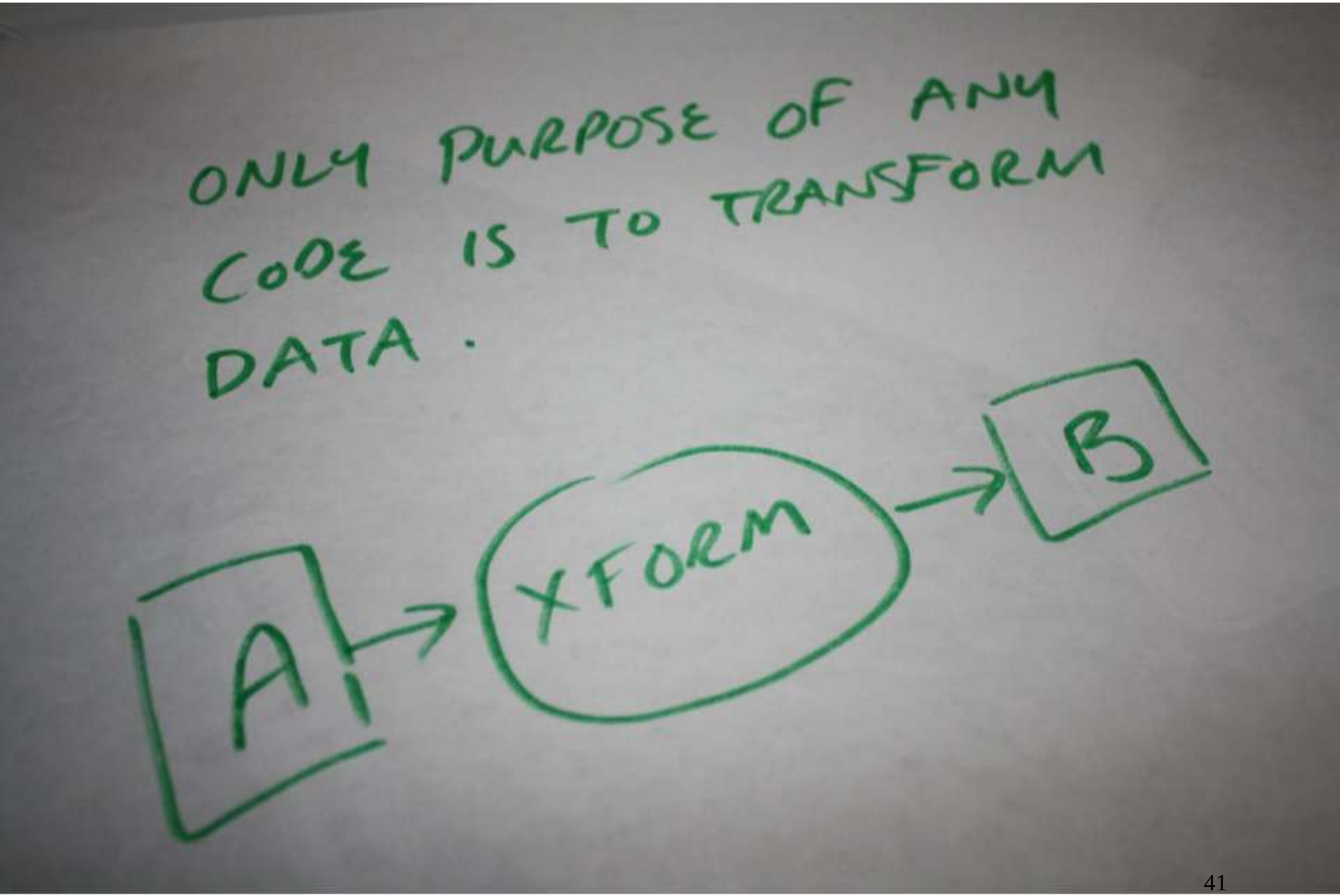

World modeling
tries to idealize the
problem.
(but you can't make a
problem simpler than it is.)


World Modeling is The
equivalent of self-help
books for programming
... solve by analogy
... solve by storytelling


World modeling introduces
new problems w/ The
Story / abstraction not
directly related to The
real problem.

LIE #3:

CODE IS MORE IMPORTANT THAN DATA.


ONLY PURPOSE OF ANY CODE IS TO TRANSFORM DATA.
A
XFORM
B

PROGRAMMER IS FUNDAMENTALLY RESPONSIBLE FOR THE DATA, NOT THE CODE.

ONLY WRITE CODE THAT
HAS DIRECT, PROVABLE VALUE.

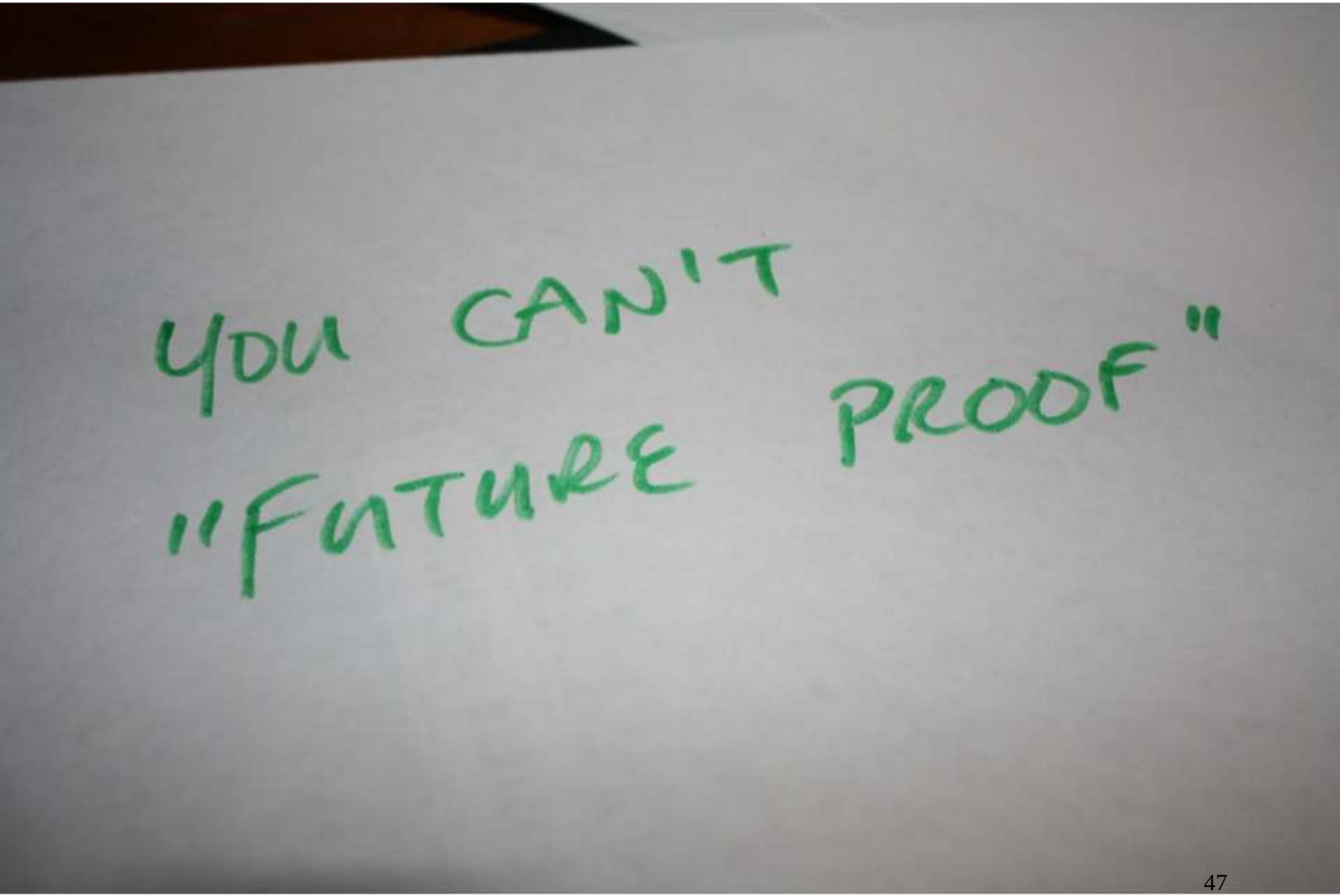
I.E. transforms DATA
IN MEANINGFUL WAY

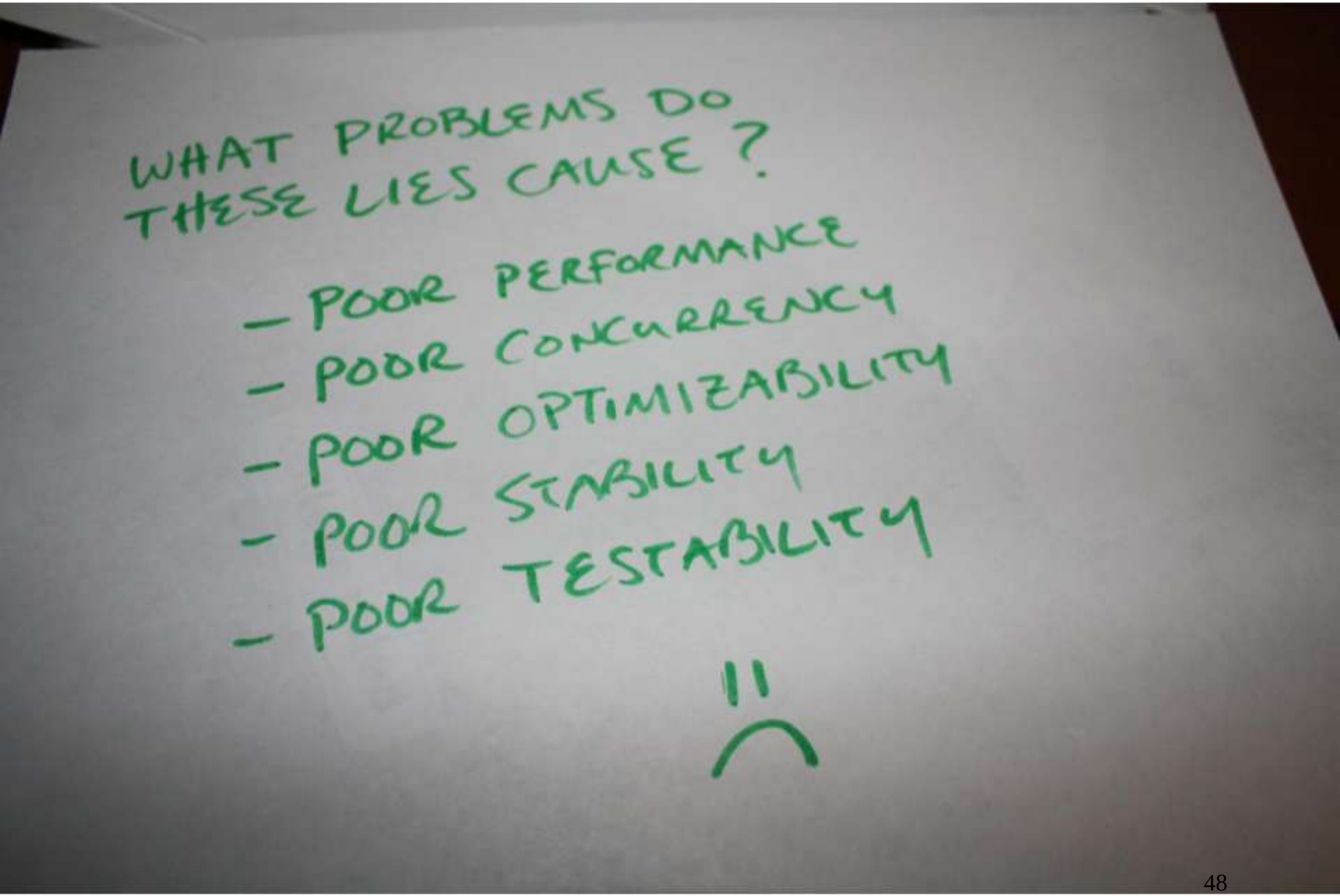
CODE IS JUST A
DATA DESCRIPTION.

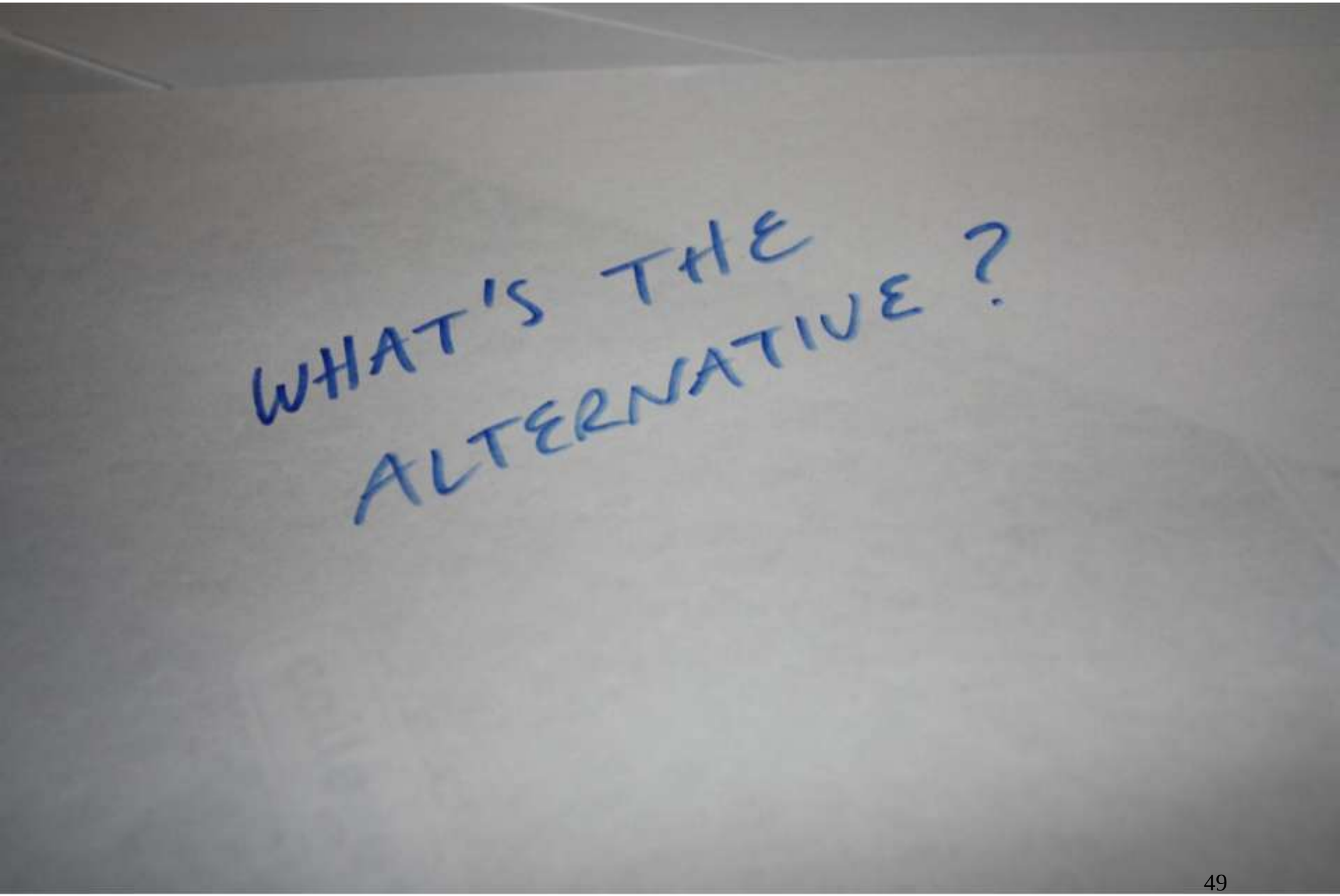
(AND IS JUST DATA.)

UNDERSTAND THE DATA
TO UNDERSTAND THE
PROBLEM.

THERE IS NO IDEAL, ABSTRACT SOLUTION TO THE PROBLEM.

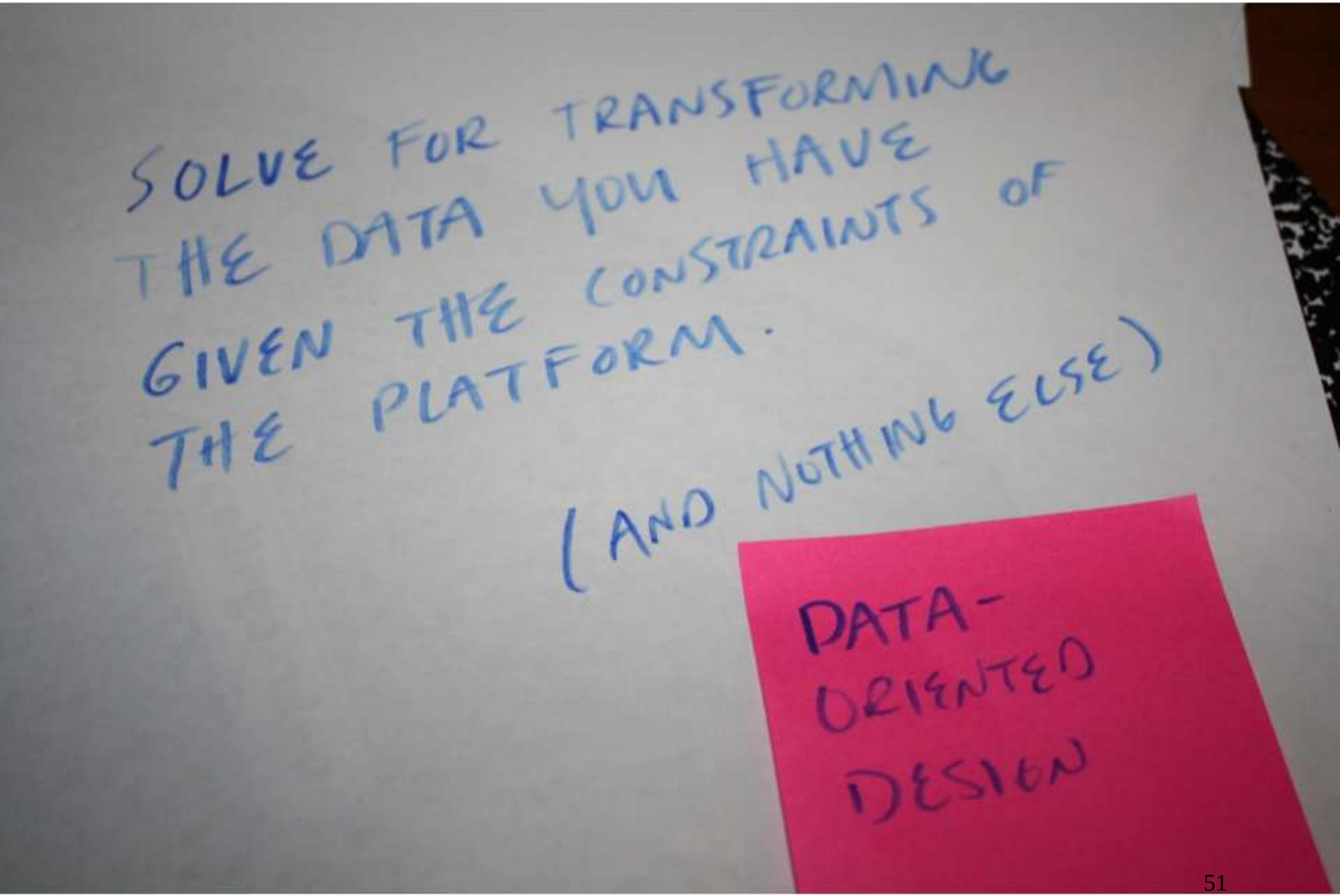

YOU CAN'T
"FUTURE PROOF"


WHAT PROBLEMS DO THESE LIES CAUSE?
- POOR PERFORMANCE
- POOR CONCURRENCY
- POOR OPTIMIZABILITY
- POOR STABILITY
- POOR TESTABILITY

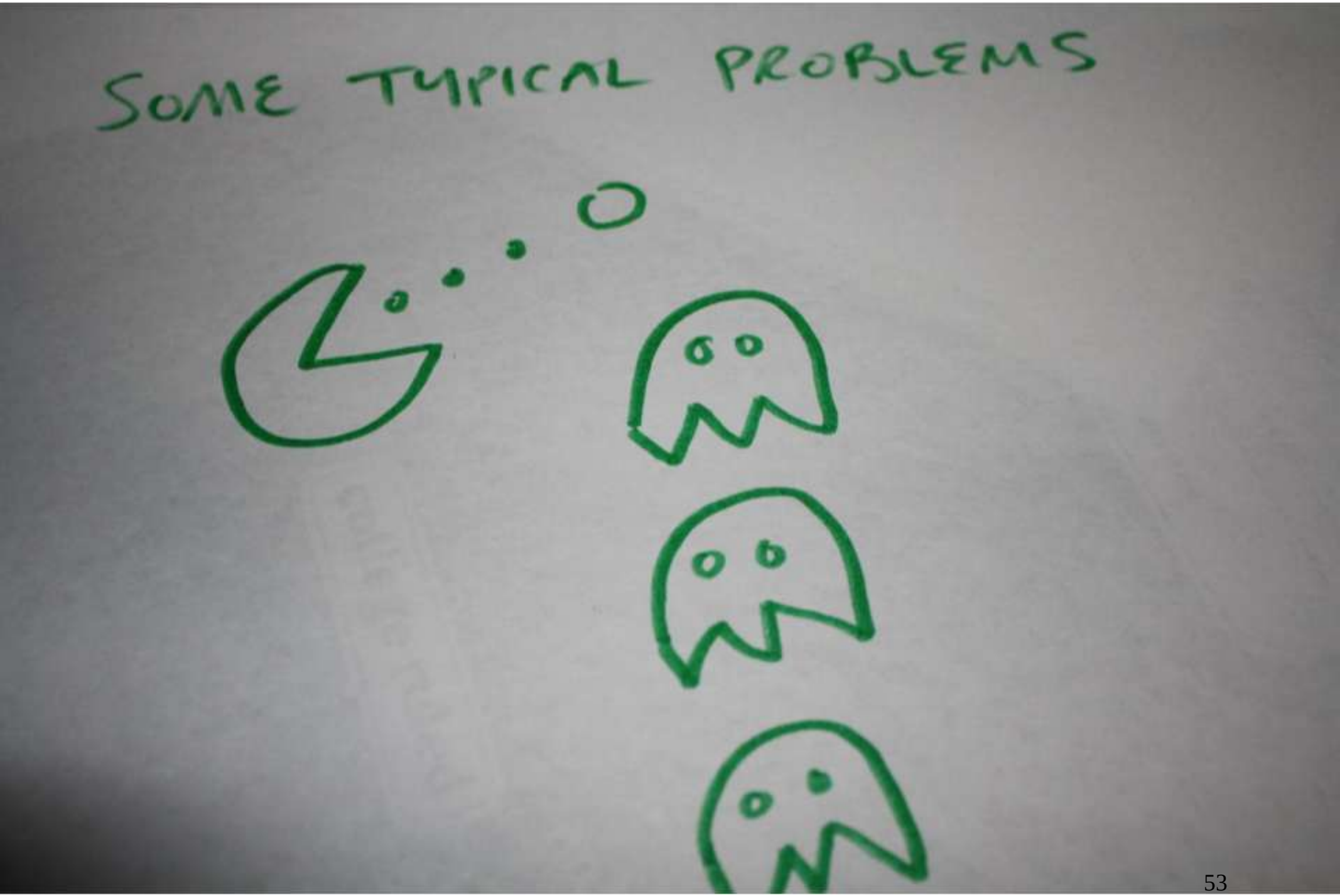

WHAT'S THE
ALTERNATIVE?

SOLVE FOR TRANSFORMING THE DATA YOU HAVE GIVEN THE CONSTRAINTS OF THE PLATFORM.

(AND NOTHING ELSE)

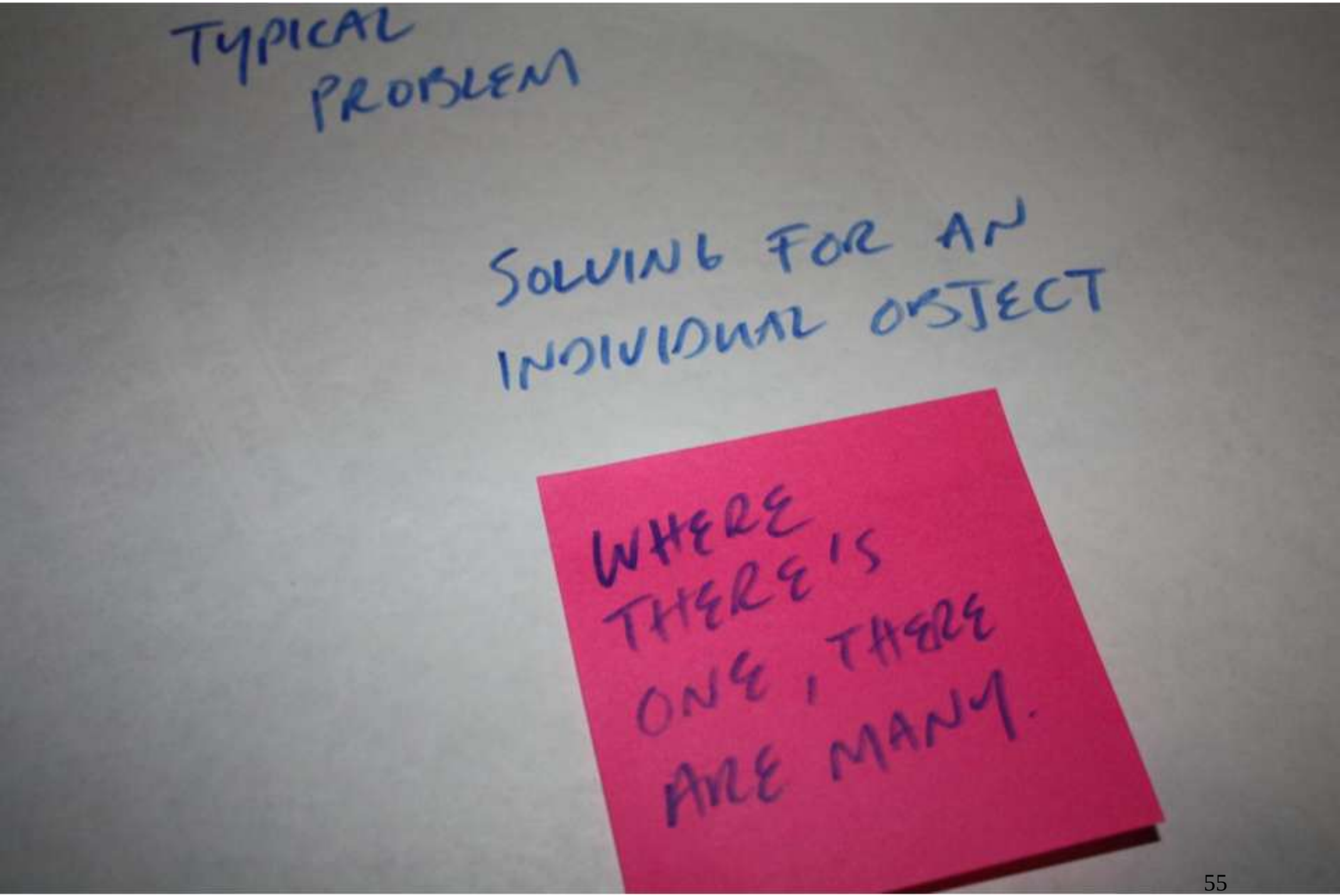

SOLVE FOR TRANSFORMING
THE DATA YOU HAVE
GIVEN THE CONSTRAINTS OF
THE PLATFORM.
(AND NOTHING ELSE)
DATA-
ORIENTED
DESIGN


PART TWO:
COMMON
ISSUES

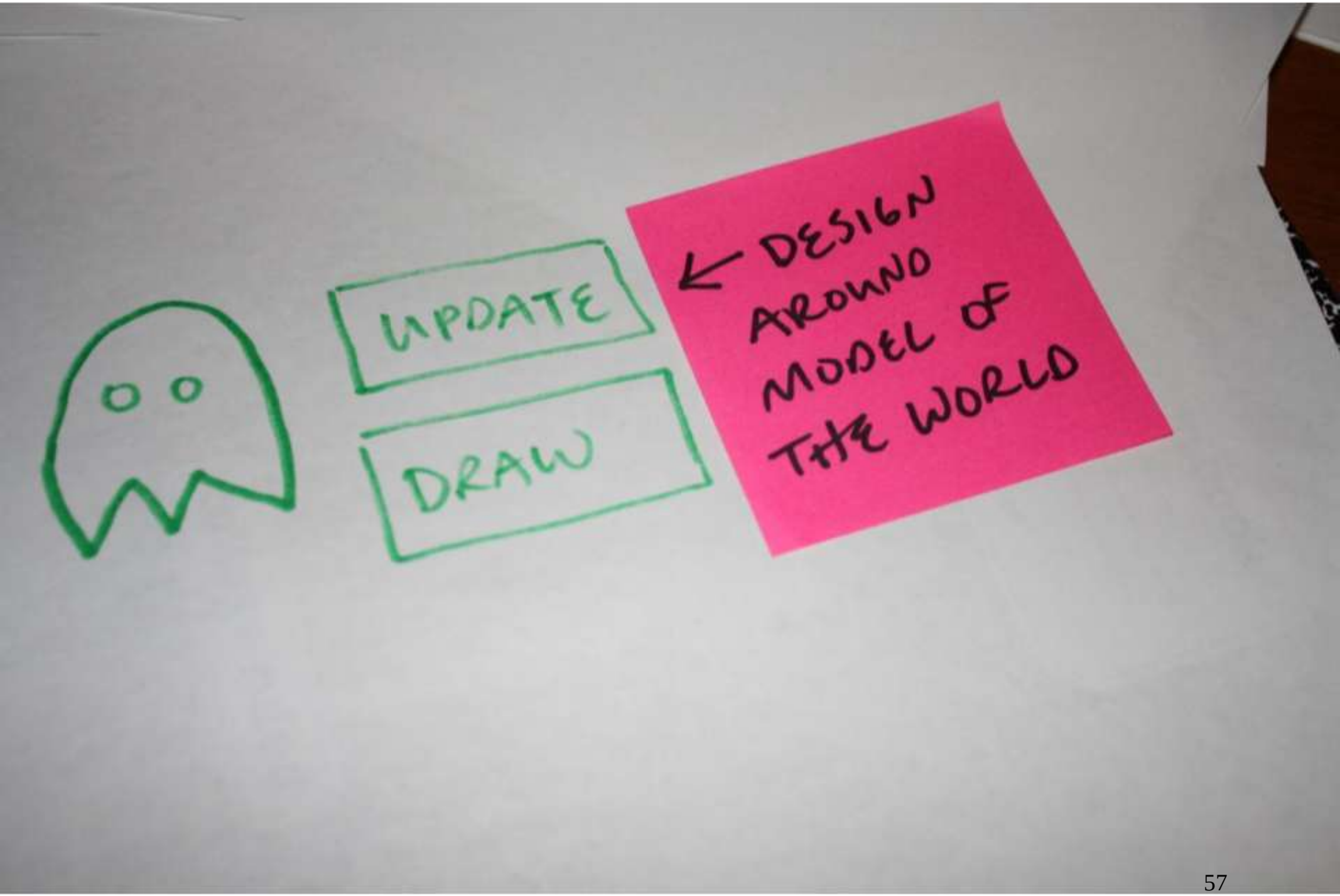

SOME TYPICAL PROBLEMS

TYPICAL PROBLEM

SOLVING FOR AN INDIVIDUAL OBJECT


TYPICAL PROBLEM
SOLVING FOR AN INDIVIDUAL OBJECT
WHERE THERE'S ONE, THERE ARE MANY.

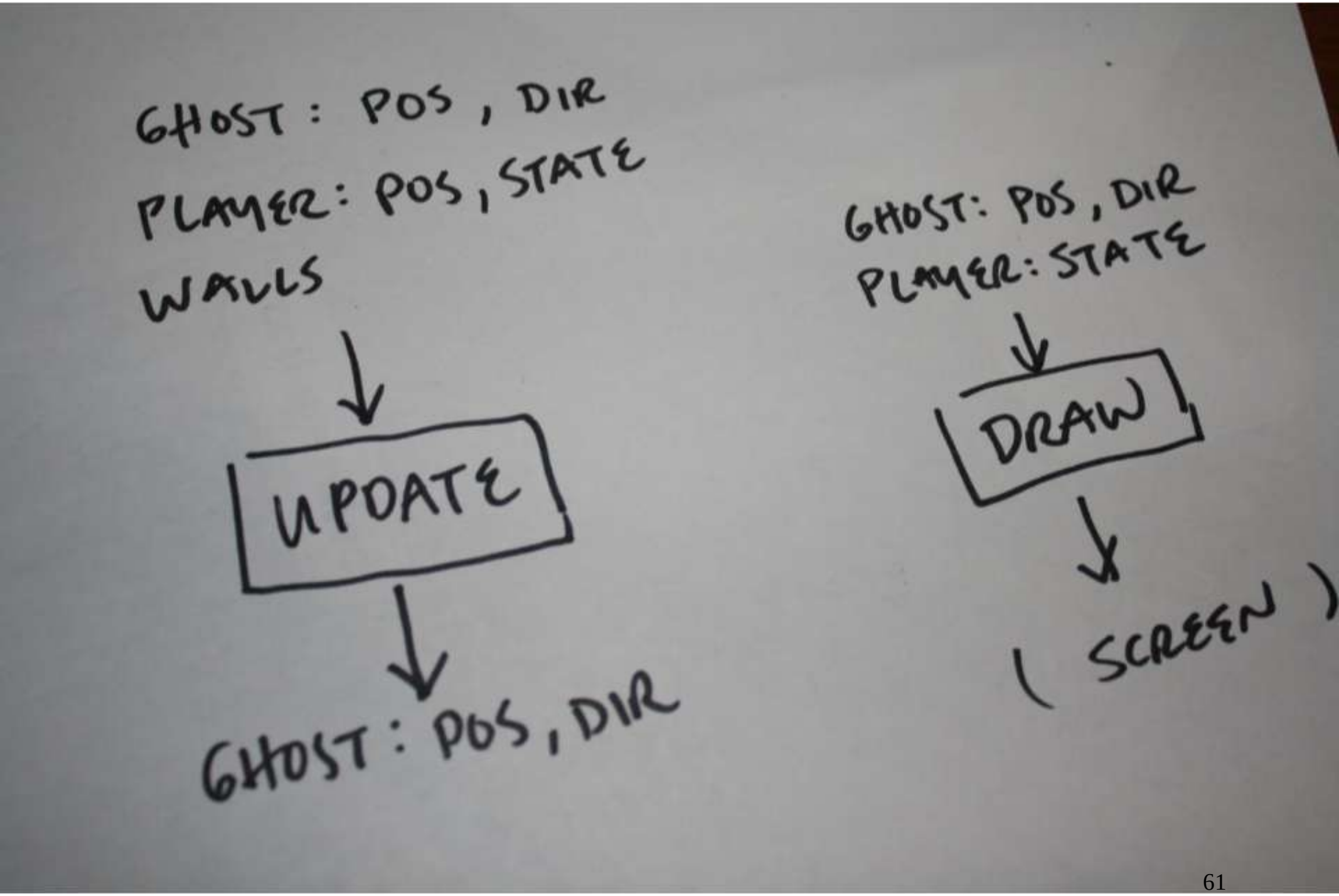

UPDATE
DRAW


UPDATE
DRAW
← DESIGN AROUND MODEL OF THE WORLD


UPDATE
DRAW
LEADS TO
STRUCTURE
EXPLOSION

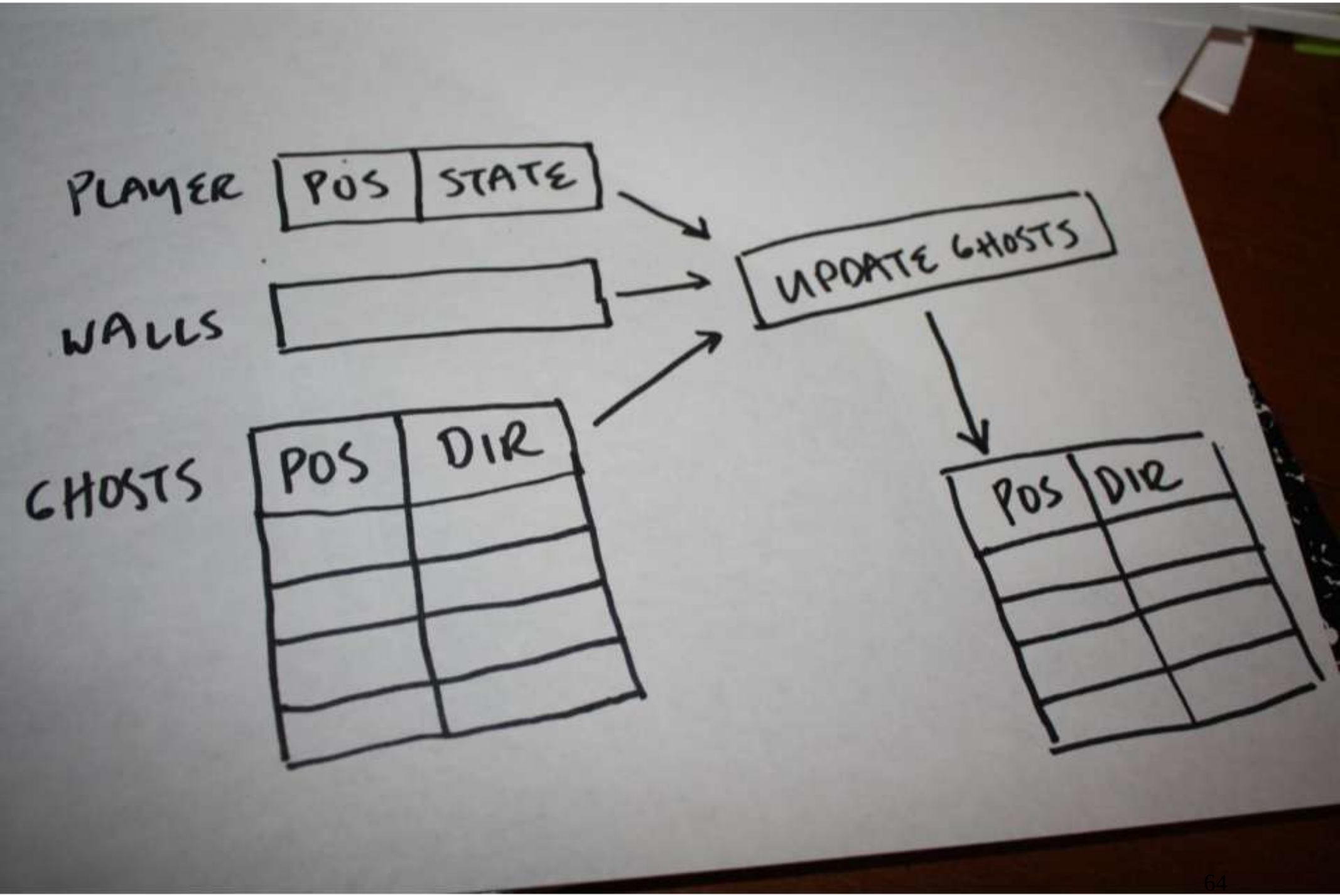

UPDATE
DRAW
DESIGN
LEADS T-
LIMITS
OPTIMIZABILITY

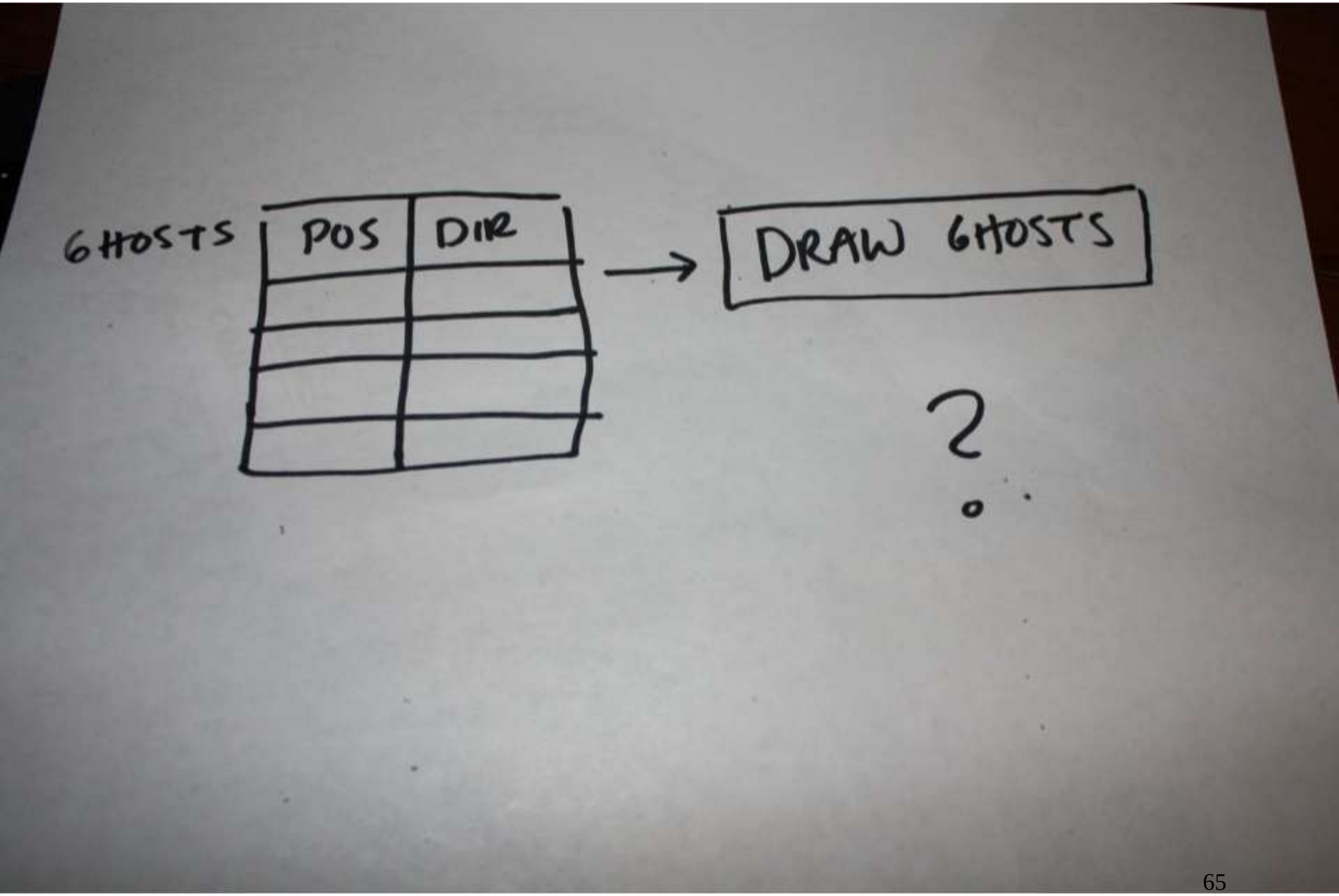

UPDATE
DRAW
UNRELATED


GHOST : POS , DIR
PLAYER : POS , STATE
WALLS
UPDATE
GHOST : POS , DIR
GHOST : POS , DIR
PLAYER : STATE
DRAW
(SCREEN)


GHOST : POS , DIR
PLAYER: POS, STATE
WALLS
UNIFORM DATA, XFORM AROUND THIS.
UPDATE

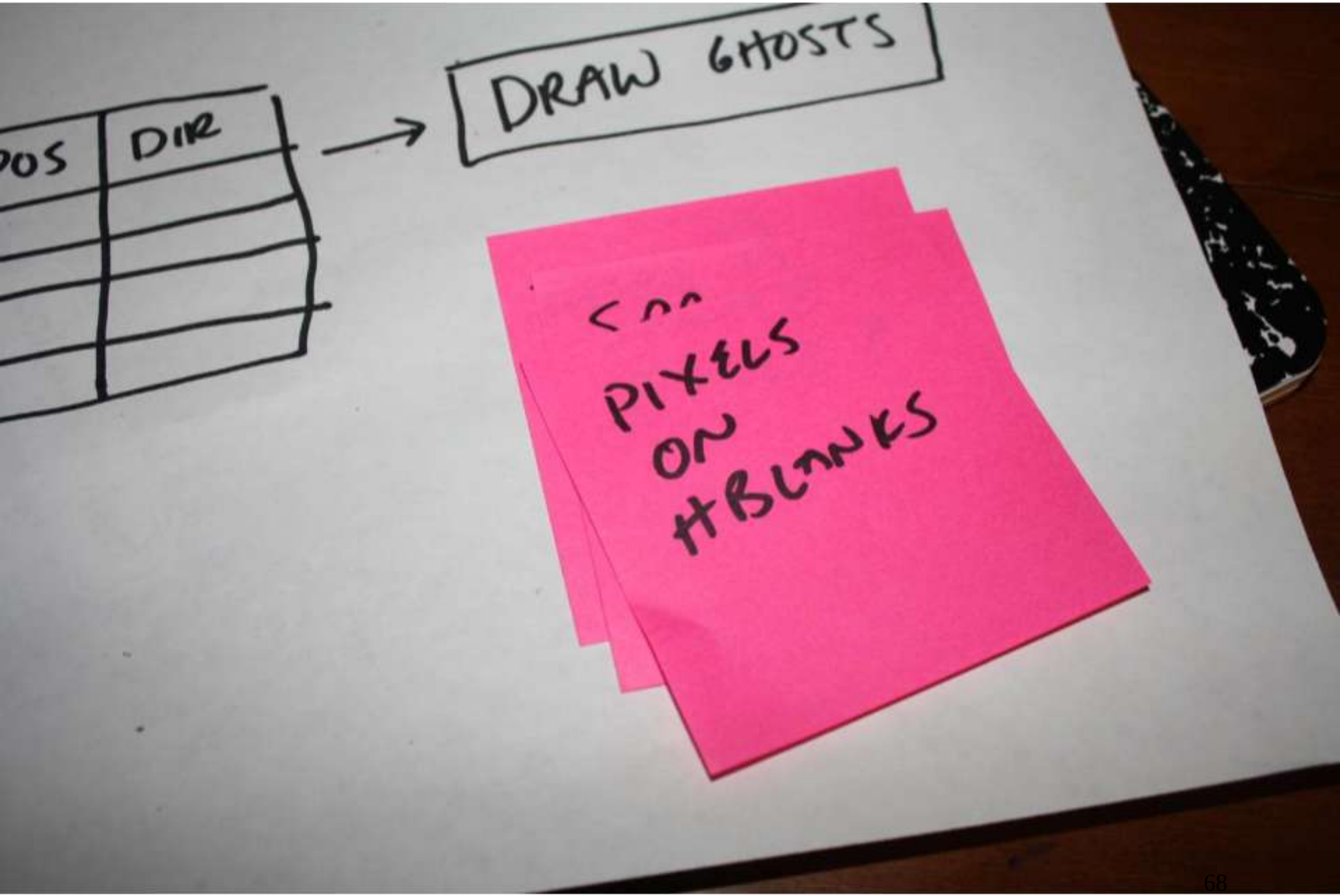

GHOST : POS , DIR
PLAYER: POS, STATE
WALLS
UPDATE
GHOST : POS, DIR
GHOST: POS , DIR
UNIFORM
DATA, XFO
AROUN
THIS.
WHY RELOAD
UNIFORM
DATA?

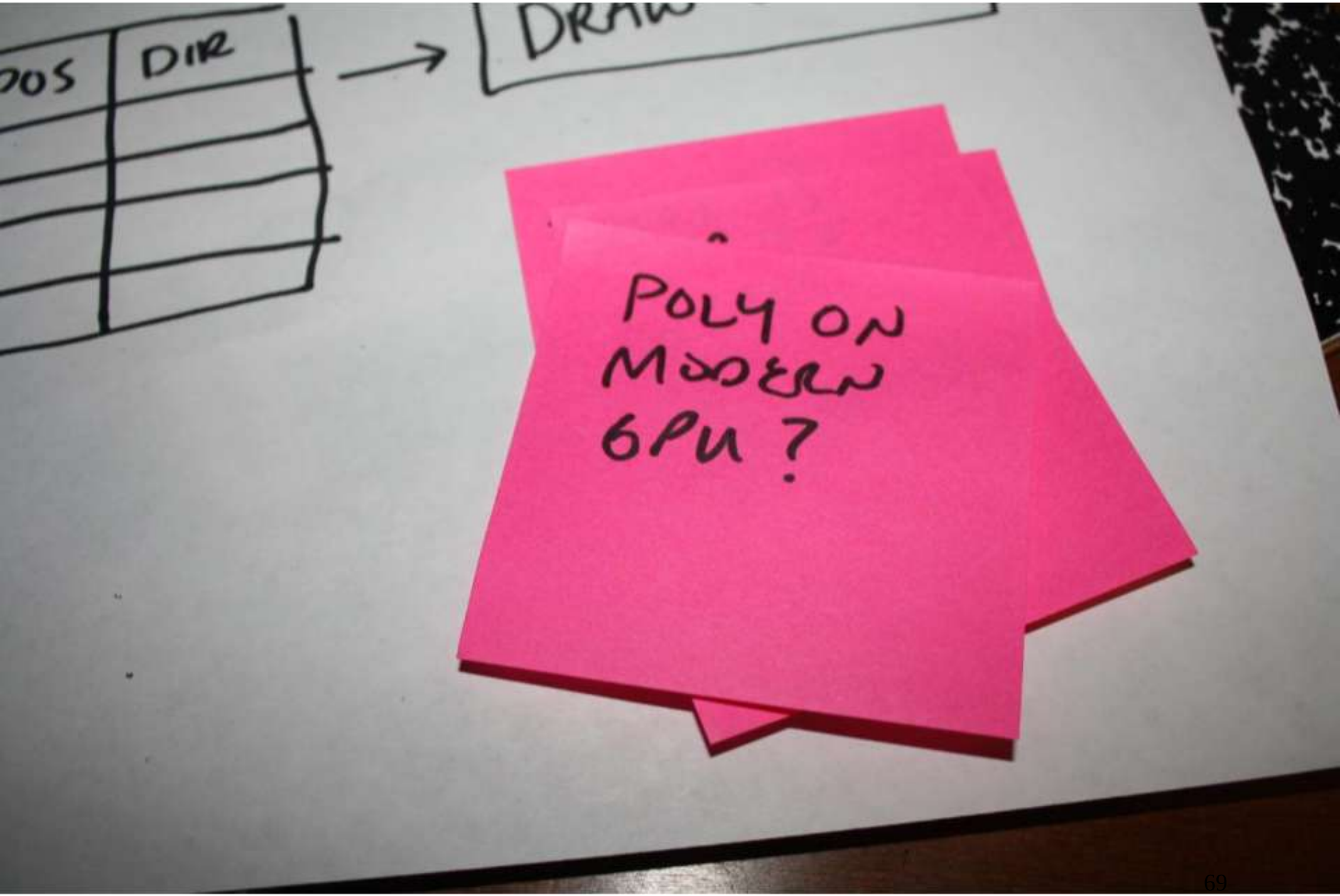

PLAYER
POS
STATE
WALLS
UPDATE GHOSTS
GHOSTS
POS
DIR
POS
DIR


GHOSTS
POS
DIR
DRAW GHOSTS
?


GHOSTS
POS
DIR
DRAW GHOSTS
THERE IS NO S/W ANSWER TO THIS

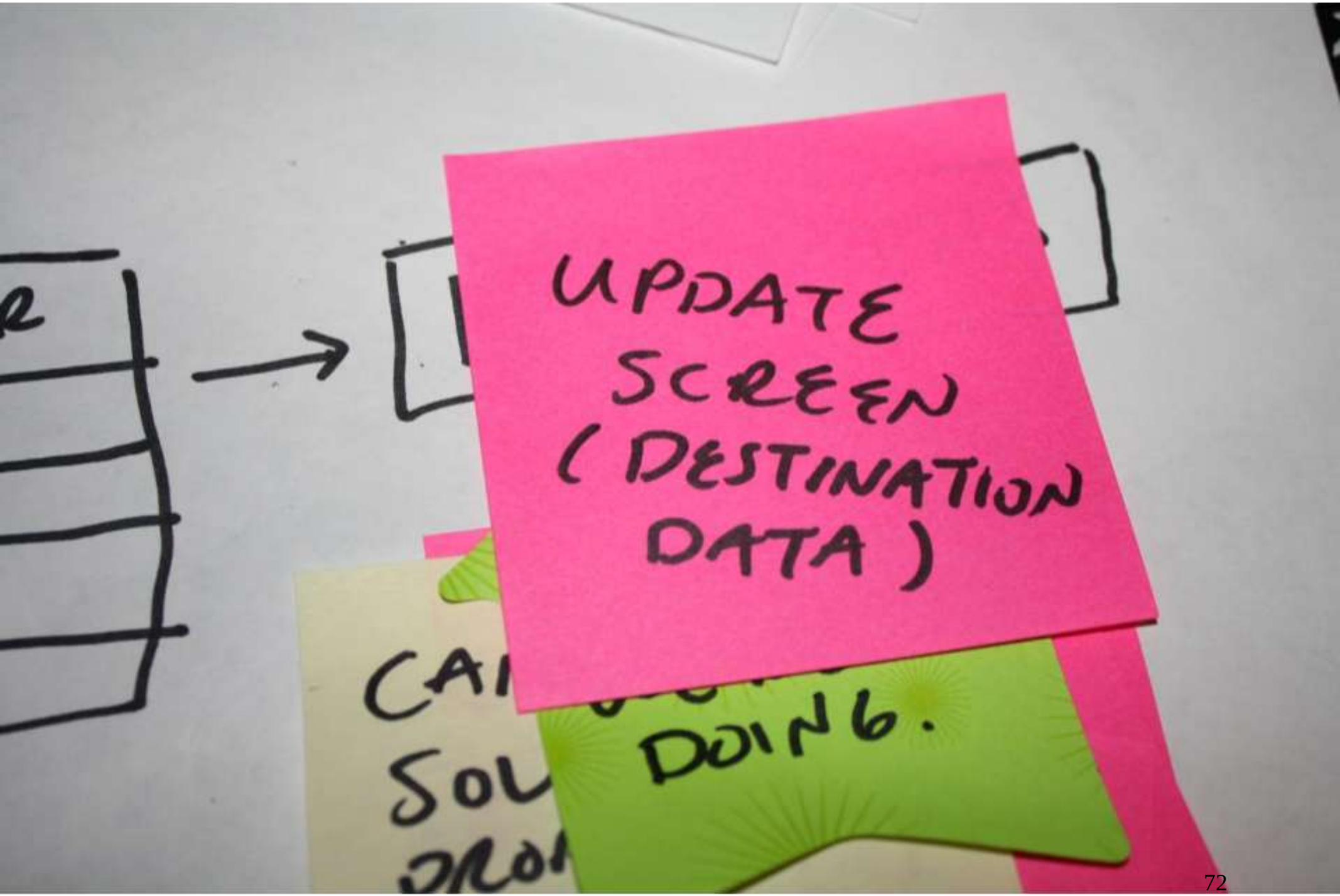

OSTS
POS
DIR
DRAW GHOSTS
SPRITE TABLE?

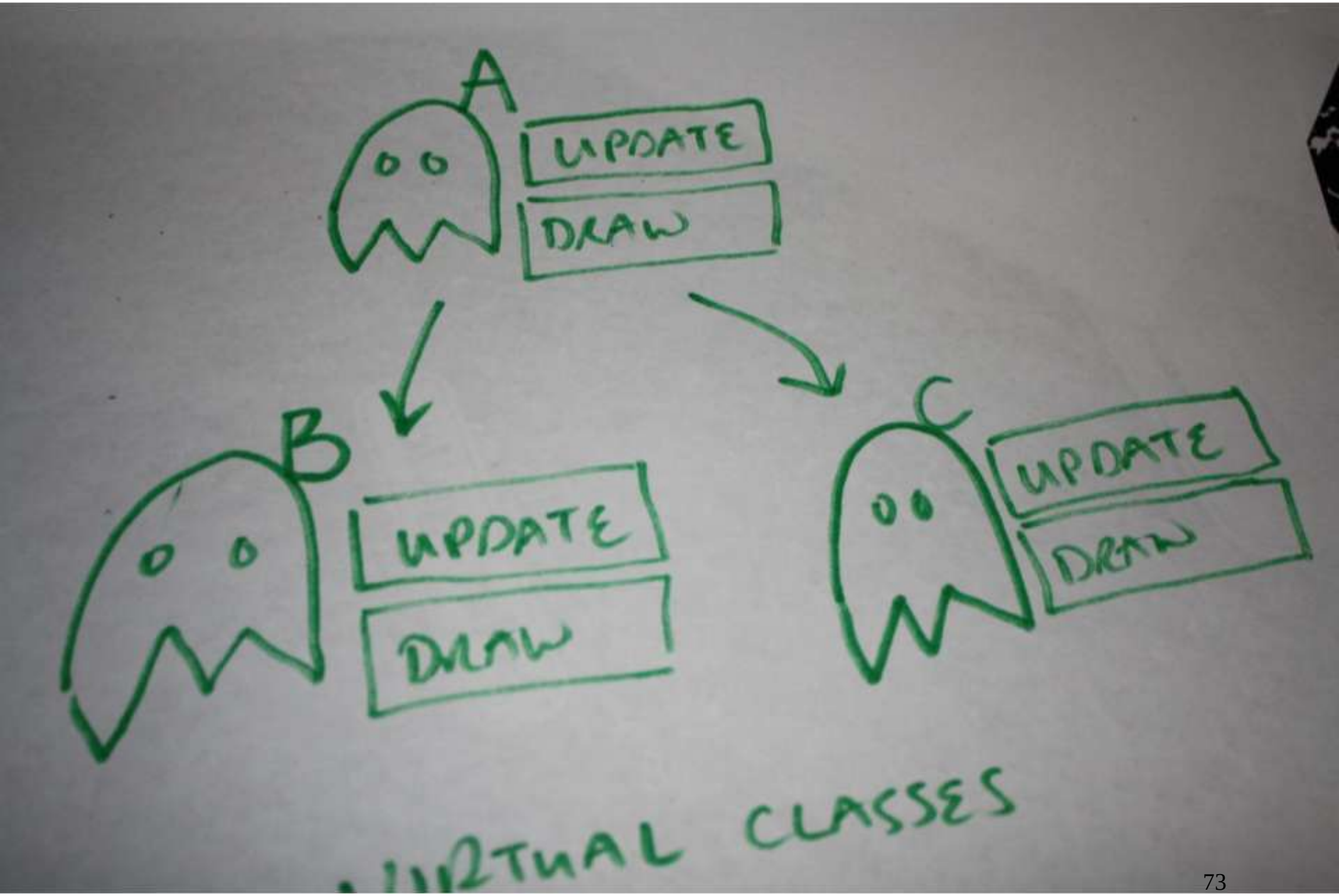

DRAW GHOSTS
DIR
<^^
PIXELS
ON
HBLANKS

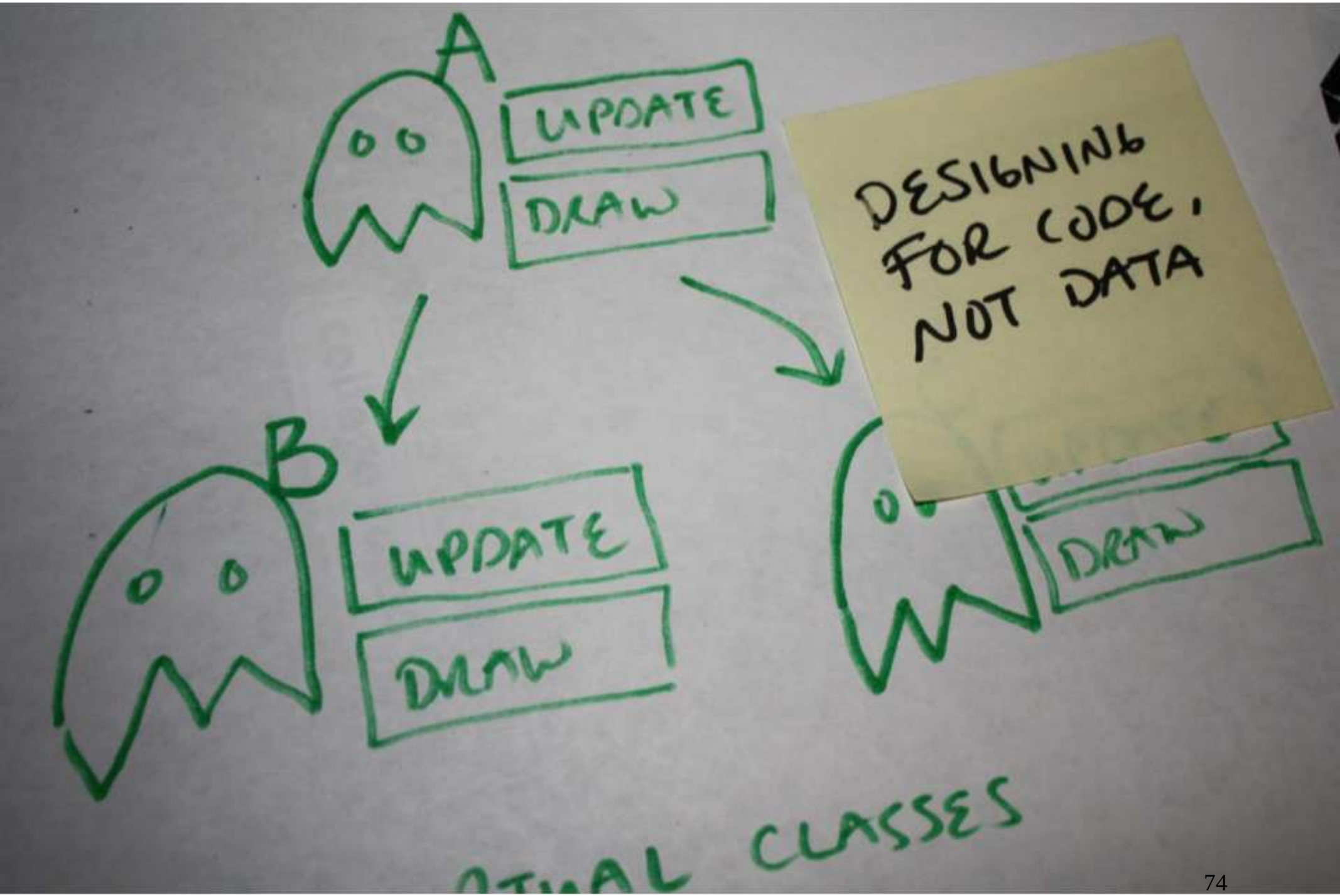

DIR
POLY ON
MODERN
GPU ?


DIR
DRAW
CAN NOT
SOLVE
PROBLEMS
W/OUT
UNDERSTANDING
PLATFORM.


DRAW GHOSTS
PLUS:
NOT WHAT
WE'RE
DOING.


UPDATE
SCREEN
(DESTINATION
DATA)
DOING.


A
UPDATE
DRAW
B
UPDATE
DRAW
C
UPDATE
DRAW
VIRTUAL CLASSES


A
UPDATE
DRAW
B
UPDATE
DRAW
DRAW
DESIGNING FOR CODE, NOT DATA
CLASSES

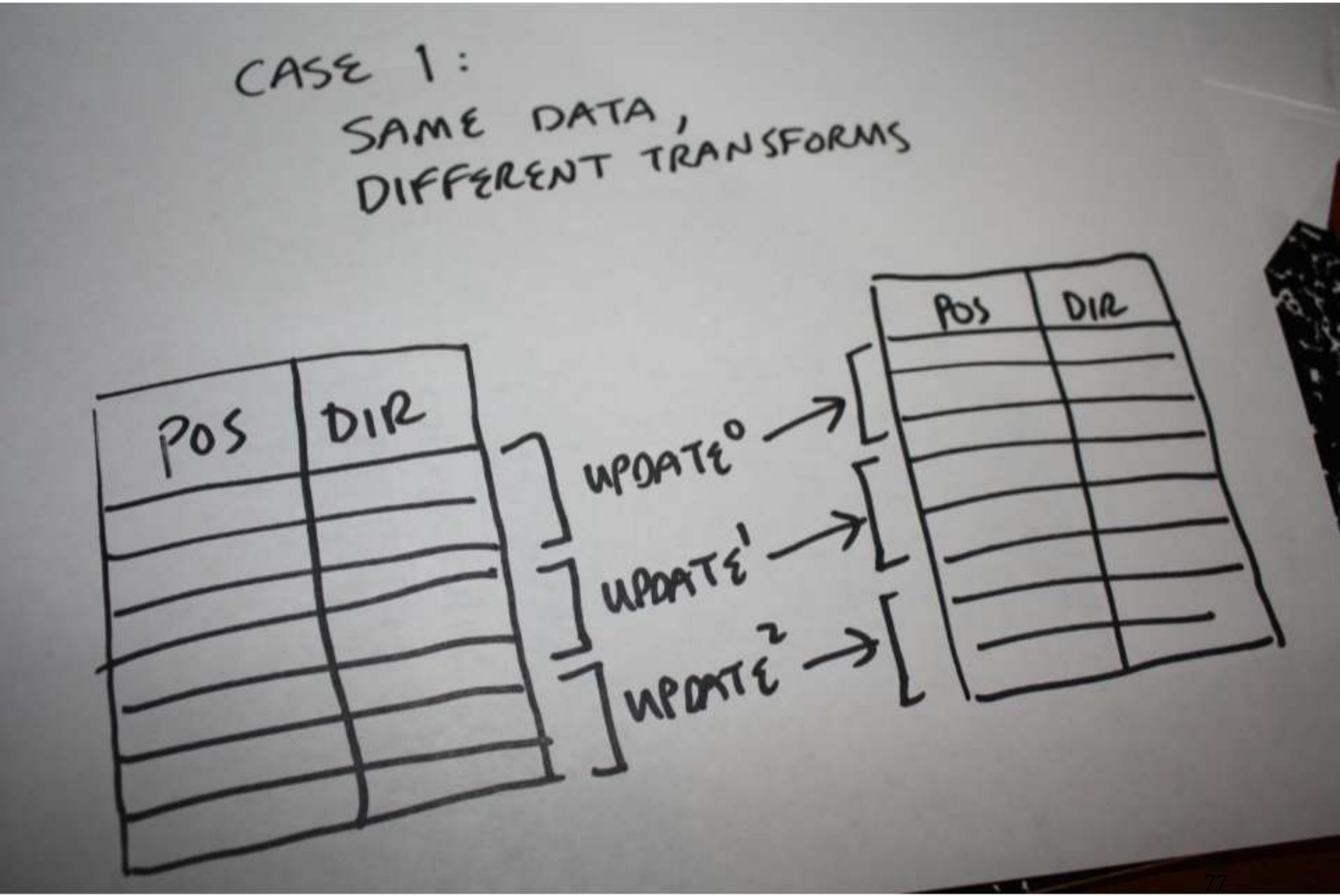
PROBLEM 1:

LOAD UNUSED DATA TO DCACHE IF GHOST IS MONOLITHIC

PROBLEM 2:

RANDOMLY RELOAD ICACHE FOR NO ADVANTAGE

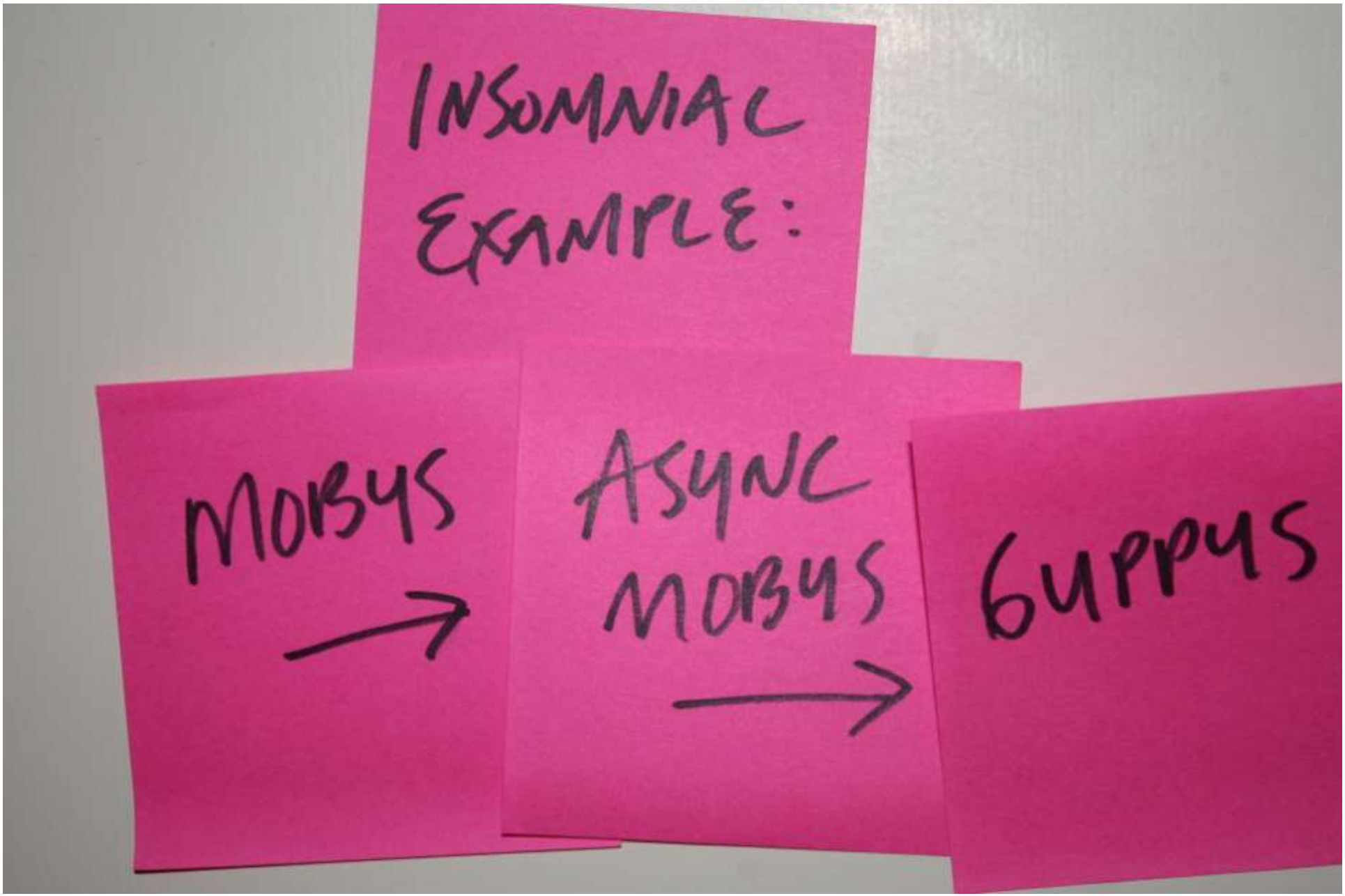
(EACH TYPE CHANGE IN LIST)


CASE 1:
SAME DATA,
DIFFERENT TRANSFORMS
POS
DIR
POS
DIR
UPDATE⁰
UPDATE¹
UPDATE²

CASE 2:

DIFFERENT DATA

(UNIQUE PROBLEMS)


INSOMNIAC
EXAMPLE:
MOBYS
ASYNC
MOBYS
GUPPYS

TYPICAL PROBLEM

GROUPING DATA
BY RELATIONSHIP
TO ALGORITHM

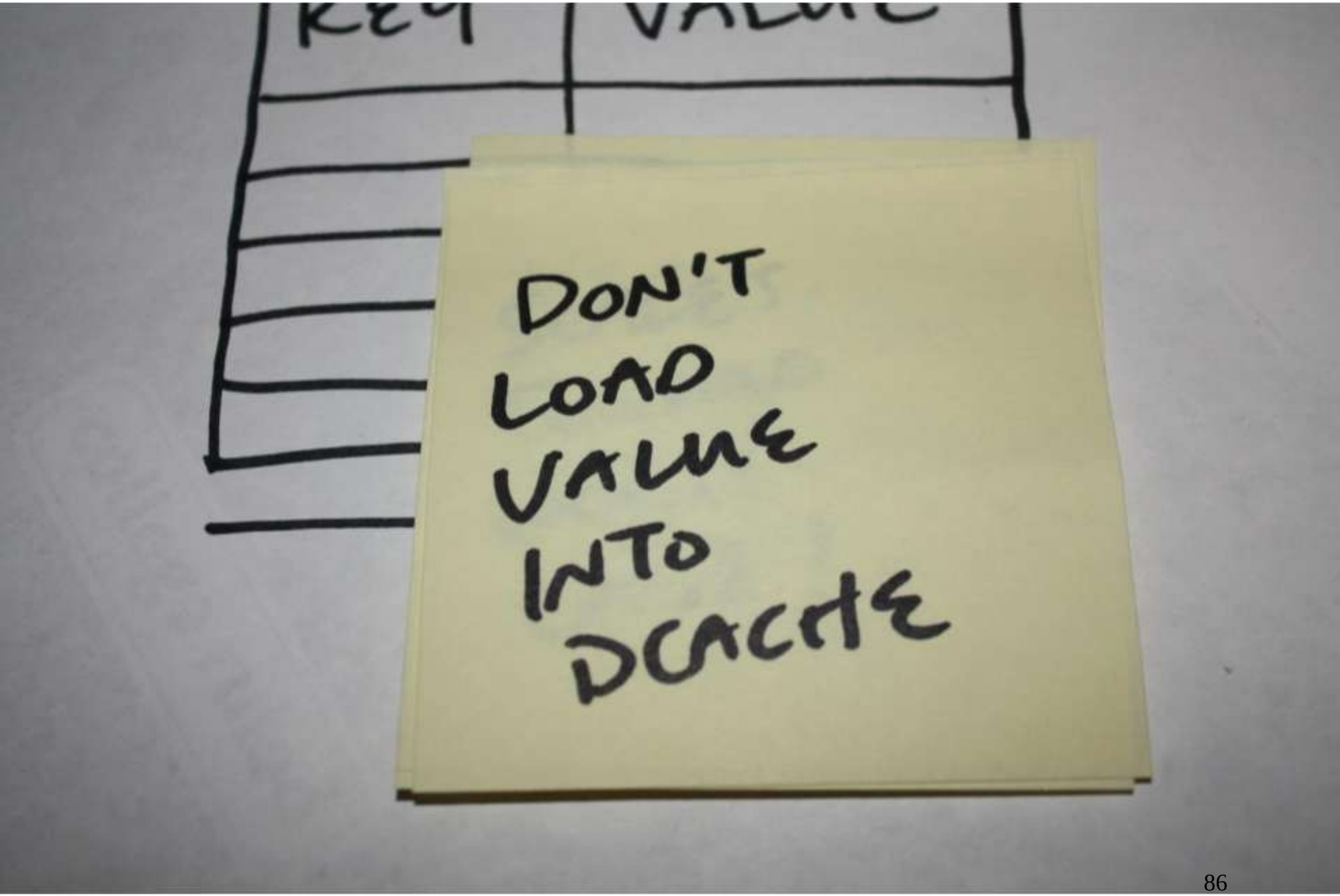
e.g. DICTIONARY LOOKUP

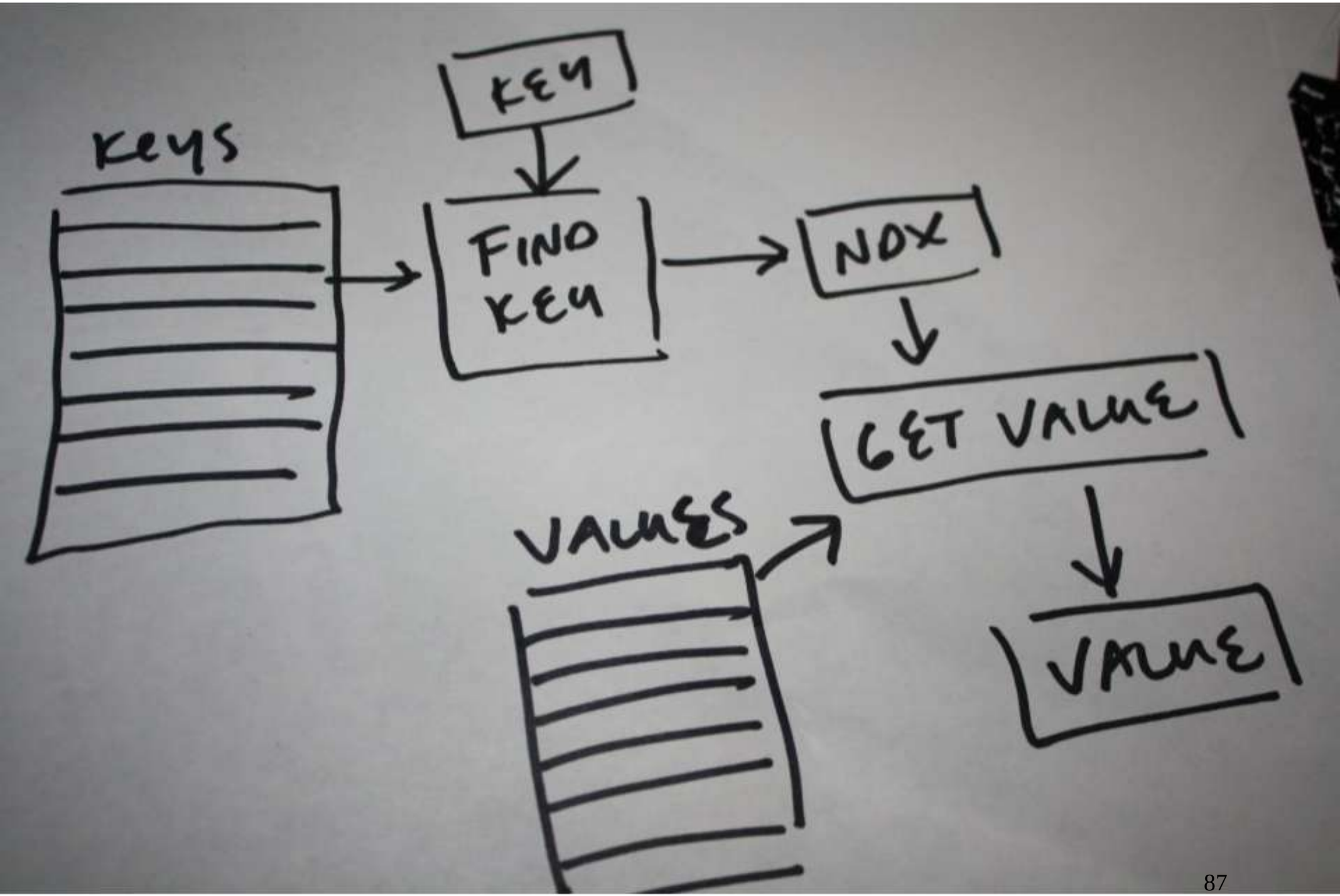

e.g. DICTIONARY LOOKUP
KEY
VALUE
DESIGNING CODE-FIRST


KEY
VALUE
WHILE SEARCH ON KEYS, WHAT IS STATISTICAL VALUE OF "VALUE"?

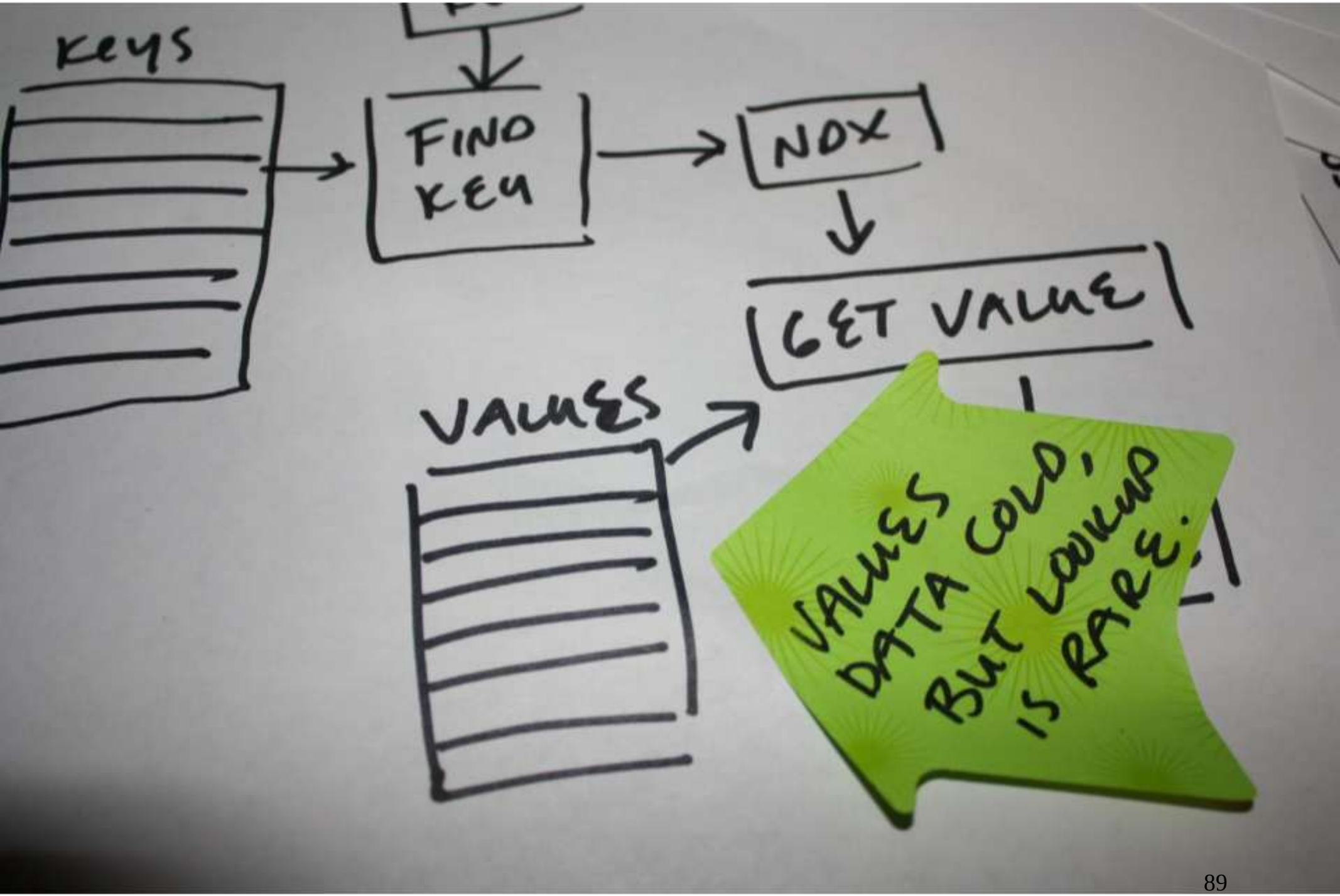

KEY
VALUE
VERY LOW!
(MORE KEYS,
LOWER
VALUE)

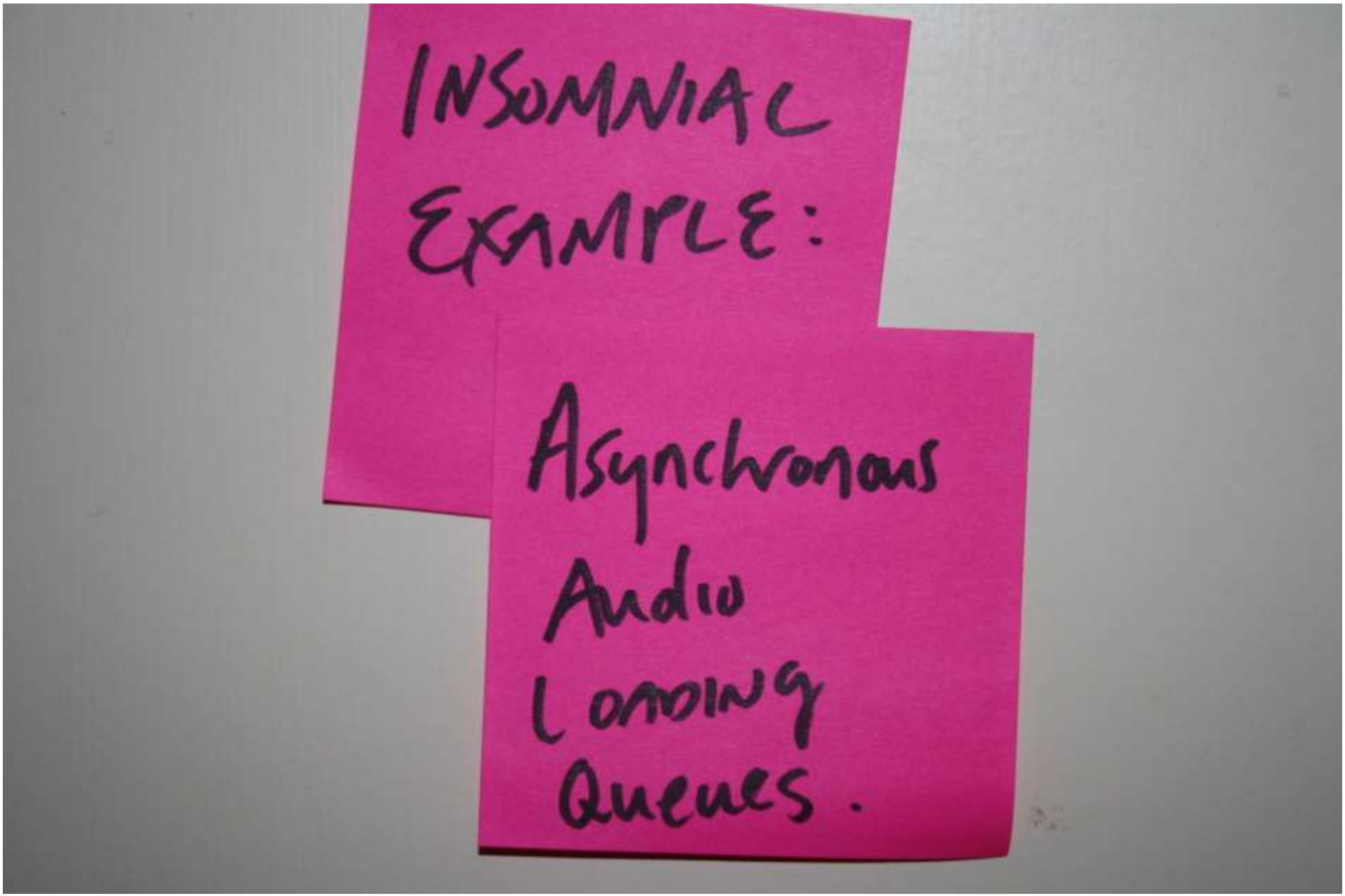

KEY
VALUE
SCALES
TOWARD
WORST-
CASE!

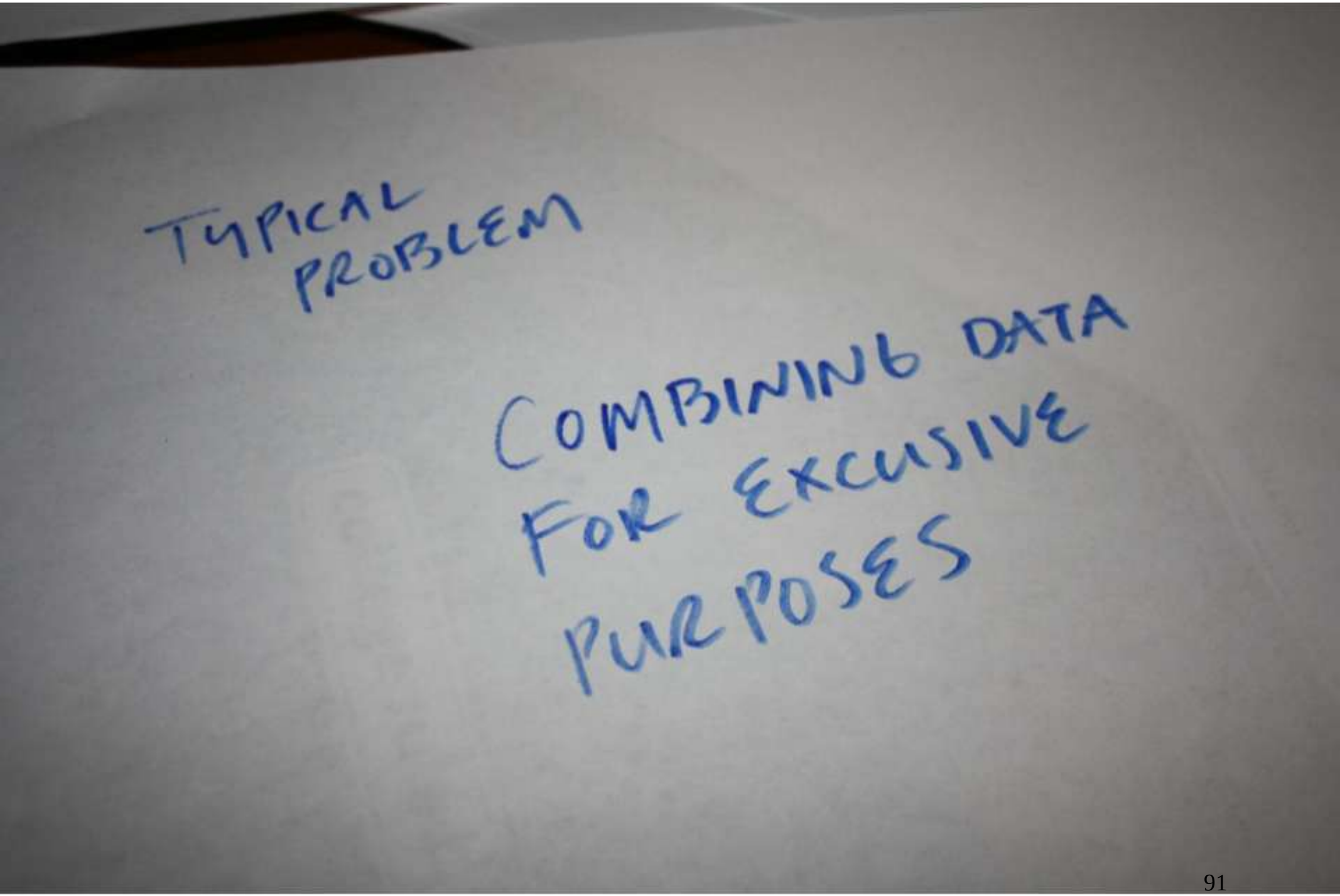

KEY
VALUE
DON'T
LOAD
VALUE
INTO
CACHE

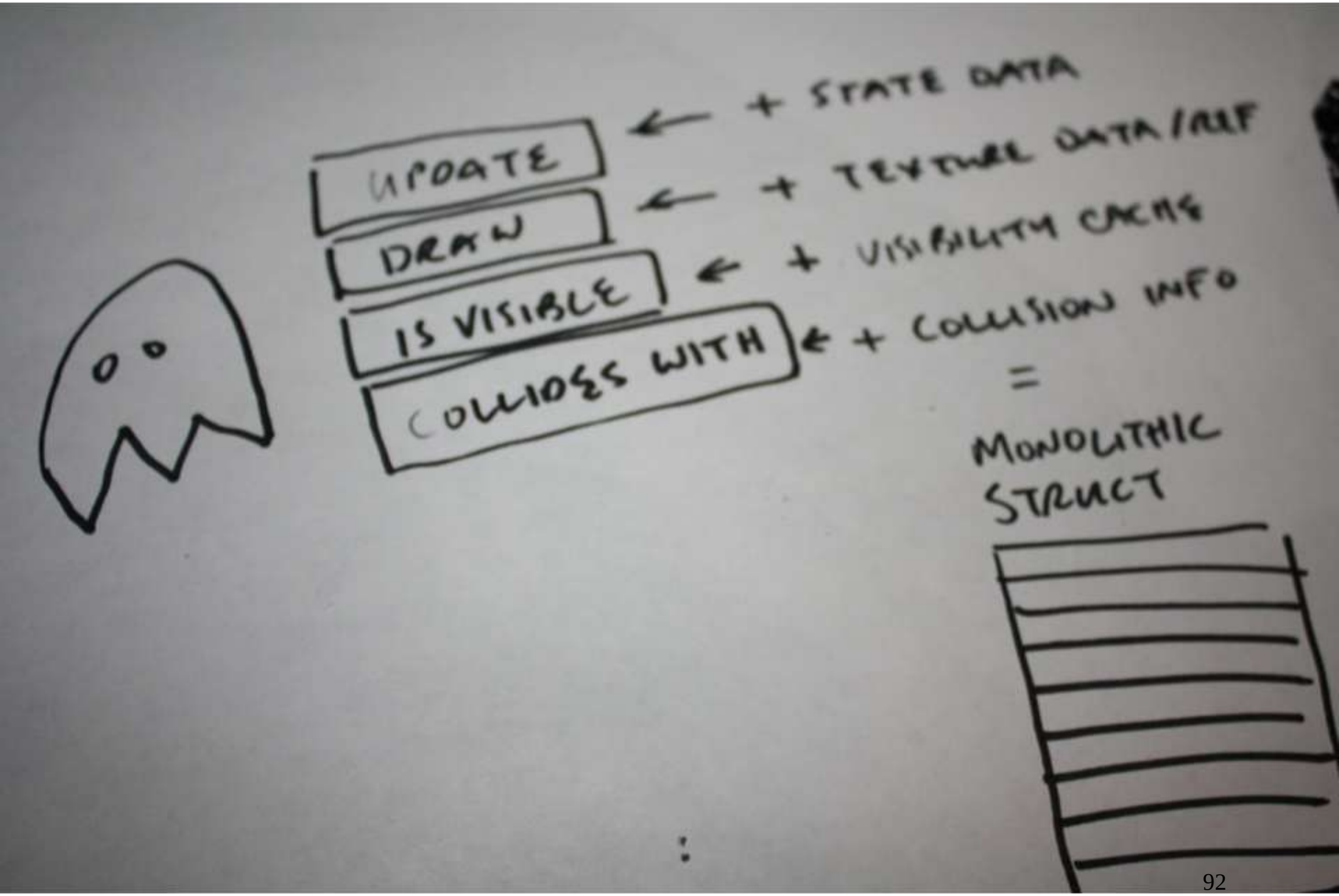

KEY
KEYS
FIND KEY
NDX
GET VALUE
VALUES
VALUE


KEY
KEYS
FIND KEY
NOX
CACHE LOADED W/ MOST LIKELY NEEDED

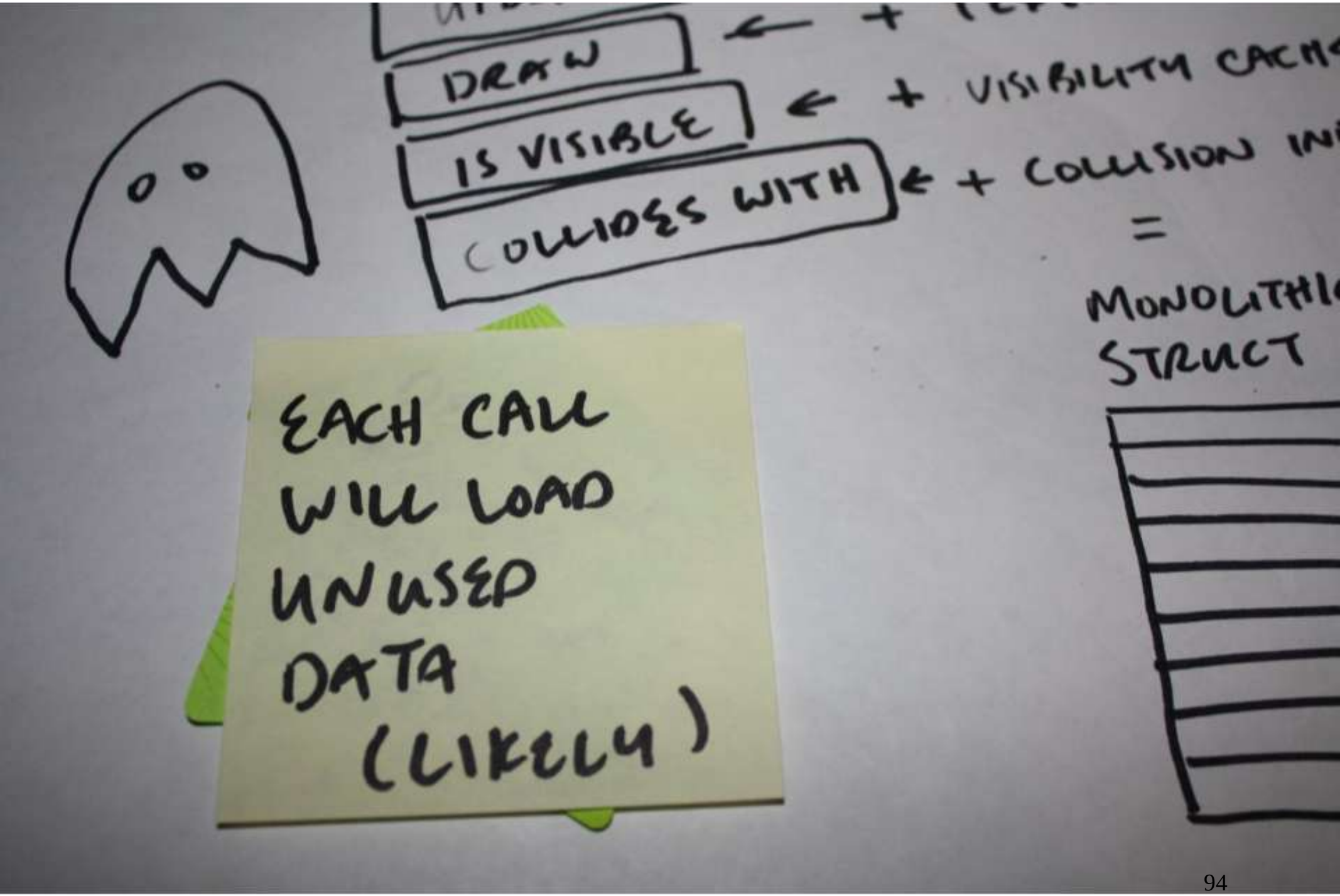

KEYS
FIND KEY
NOX
GET VALUE
VALUES
VALUES DATA COLD, BUT LOOKUP IS RARE.


INSOMNIAC
EXAMPLE:
Asynchronous
Audio
Loading
Queues.


TYPICAL PROBLEM
COMBINING DATA FOR EXCLUSIVE PURPOSES

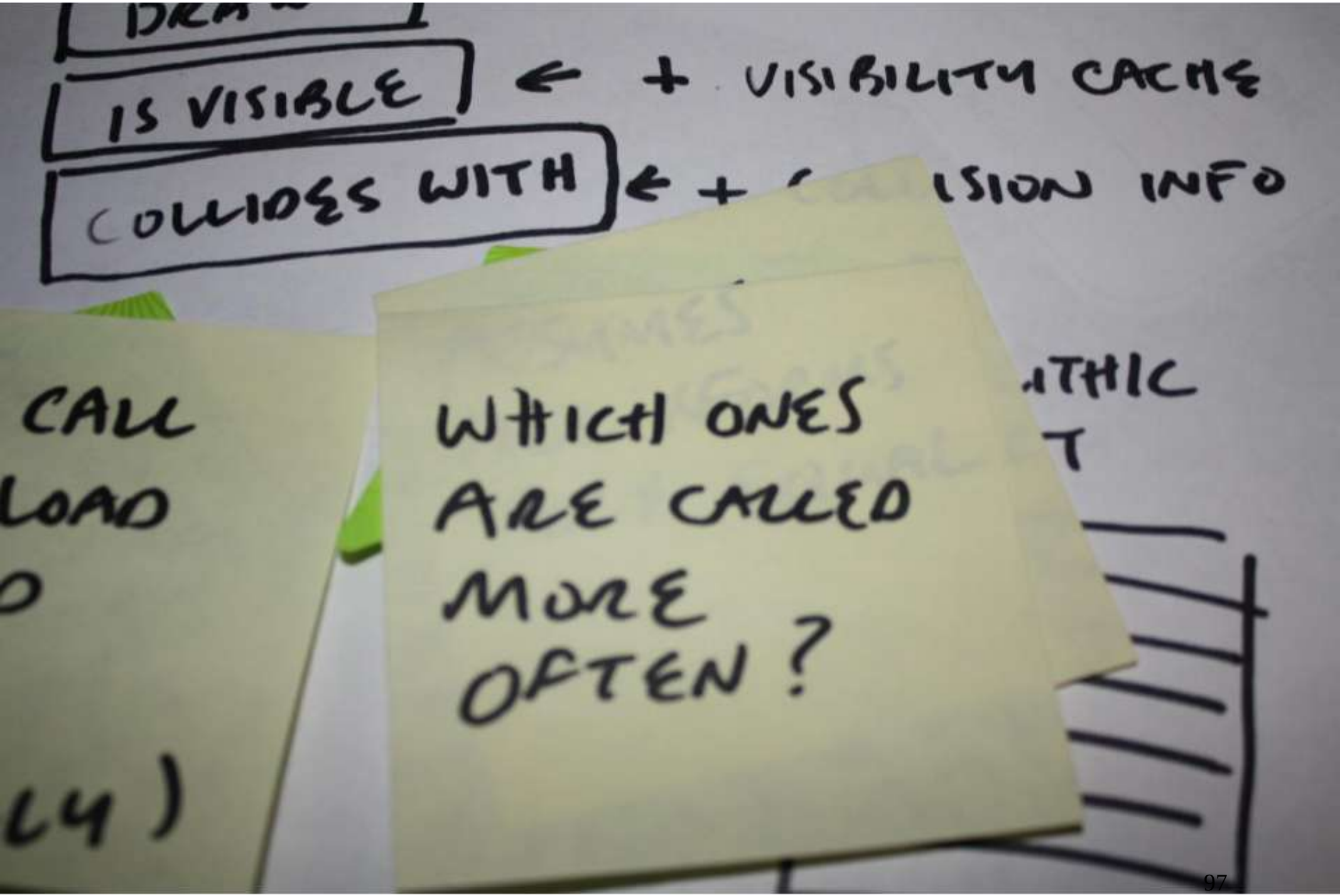

UPDATE
DRAW
IS VISIBLE
COLLIDES WITH
+ STATE DATA
+ TEXTURE DATA/REF
+ VISIBILITY CACHE
+ COLLISION INFO
=
MONOLITHIC
STRUCT

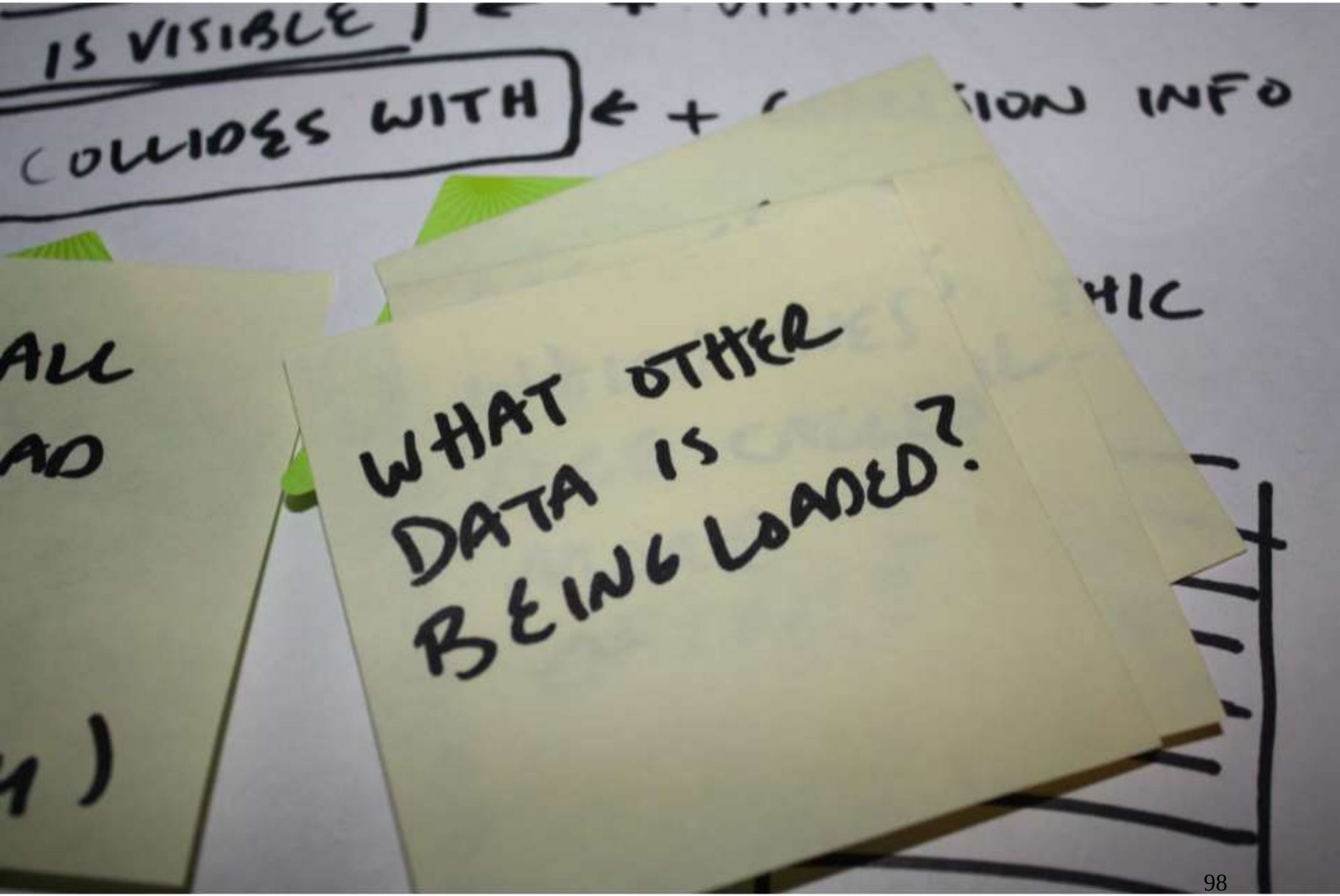

UPDATE
DRAW
IS VISIBLE
COLLIDES WITH
+ STATE DATA
+ TEXTURE DATA/REF
+ VISIBILITY CACHE
+ COLLISION INFO
=
MONOLITHIC STRUCT
DESIGNING AROUND MODEL OF WORLD


DRAW
IS VISIBLE
COLLIDES WITH
+ VISIBILITY CACH
+ COLLISION IN
=
MONOLITHI
STRUCT
EACH CALL
WILL LOAD
UNUSED
DATA
(LIKELY)


UPDATE
DRAW
IS VISIBLE
COLLIDES WITH
+ STATE DATA
+ TEXTURE DATA
+ VISIBILITY CACH
+ COLLISION
=
EACH CALL
WILL LOAD
DESIGNING
CODE
FIRST


UPDATE
DRAW
IS VISIBLE
COLLIDES WITH
+ TEXTURE DATA/REF
+ VISIBILITY CACHE
+ COLLISION INFO
EACH CALL
WILL LOAD
UNUSED
DATA
ASSUMES
ALL XFORMS
ARE EQUAL


IS VISIBLE
+ VISIBILITY CACHE
COLLIDES WITH
WHICH ONES ARE CALLED MORE OFTEN?

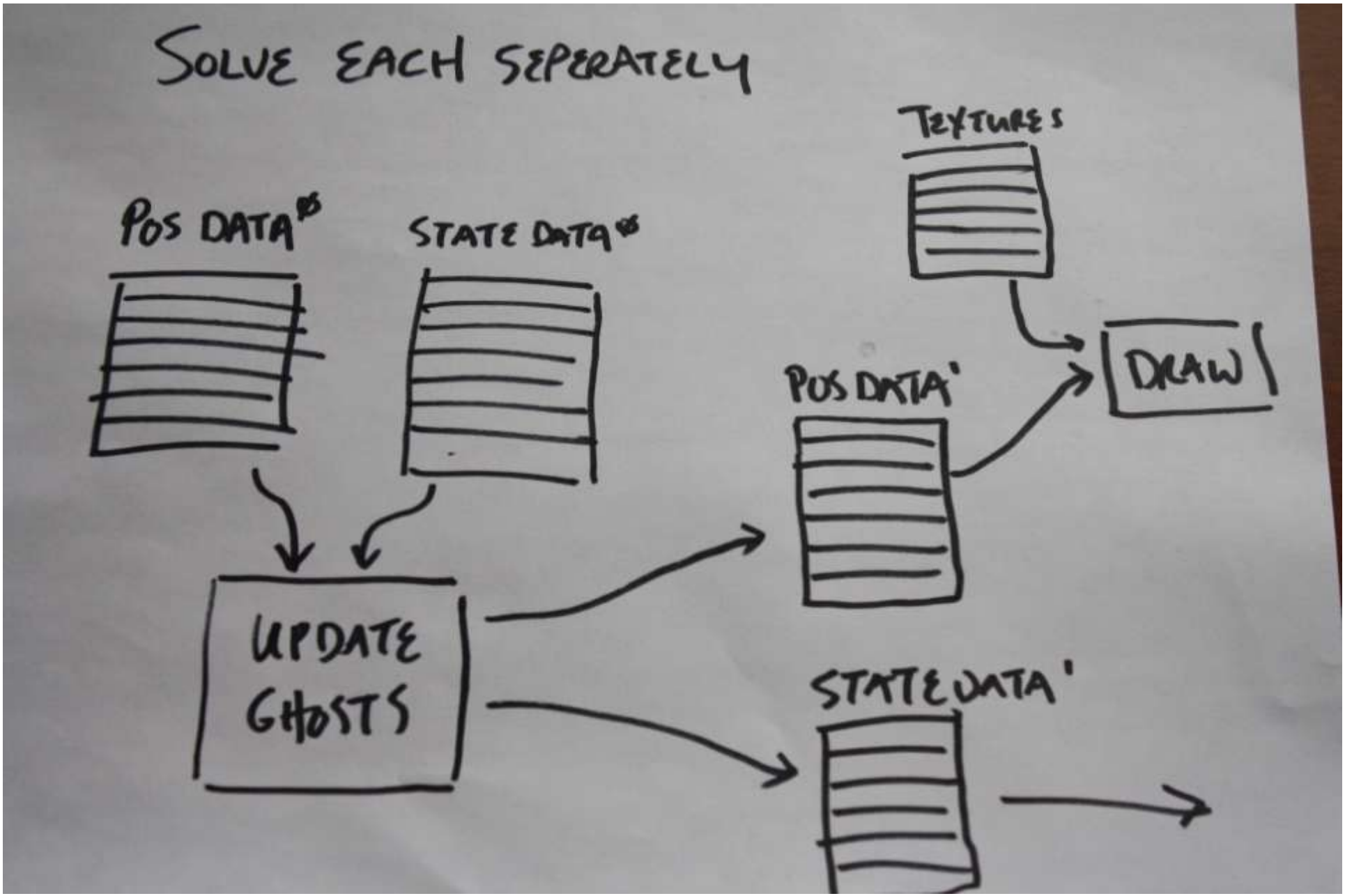

IS VISIBLE
COLLIDES WITH
WHAT OTHER DATA IS BEING LOADED?

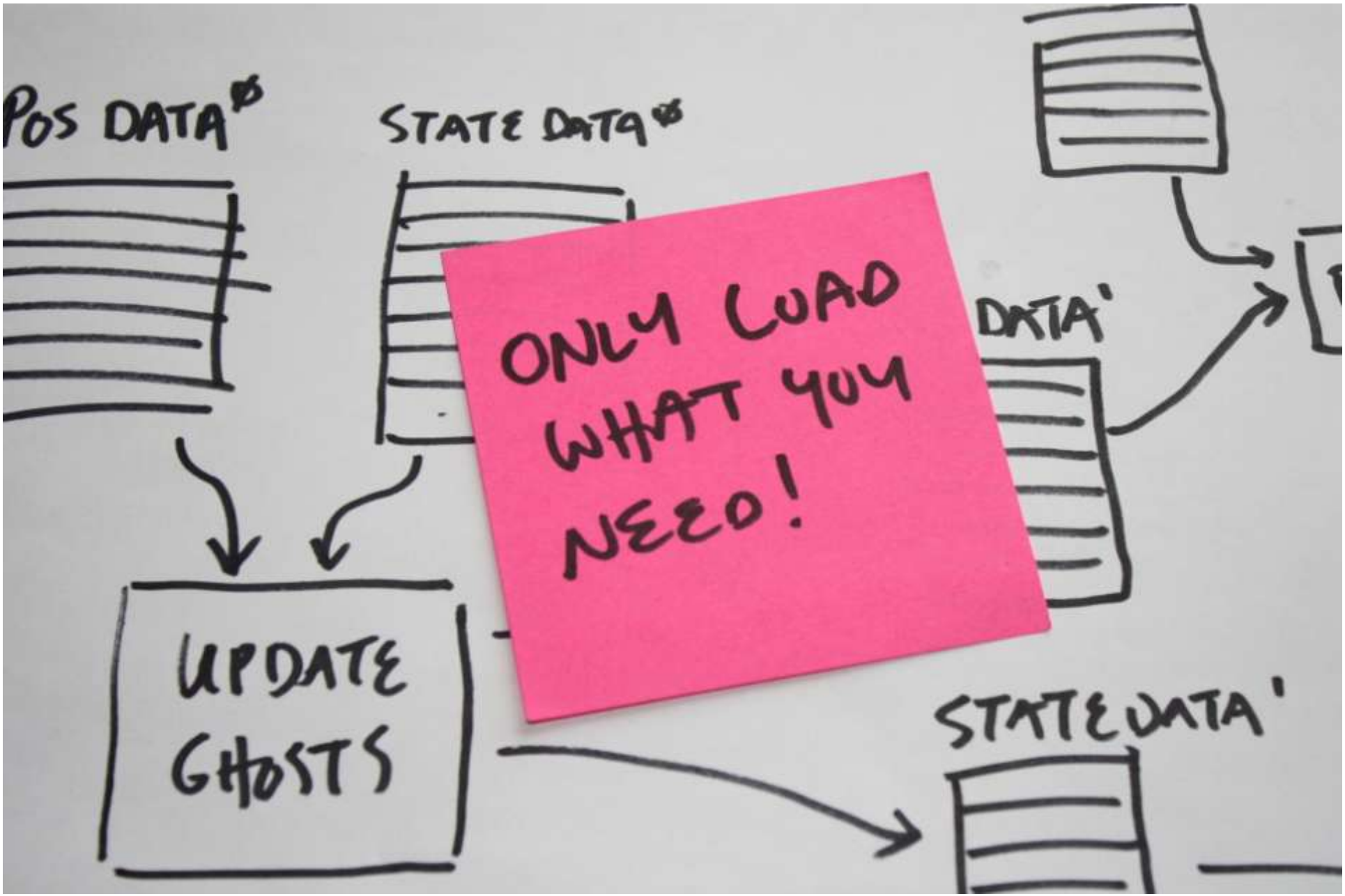
DIFFERENT DATA

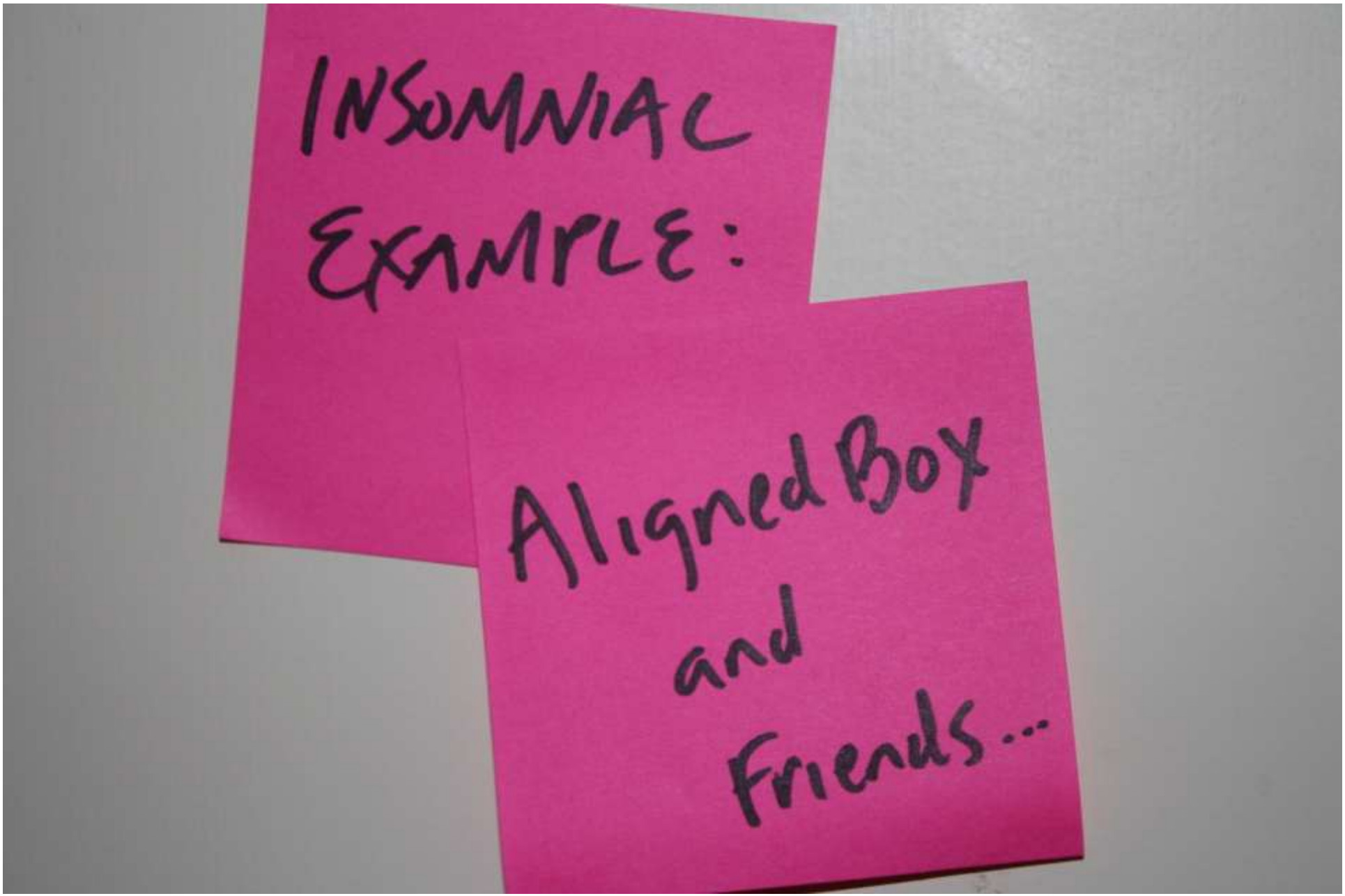
MAKES DIFFERENT

PROBLEM.

DIFFERENT PROBLEM
REQUIRES DIFFERENT
SOLUTION.

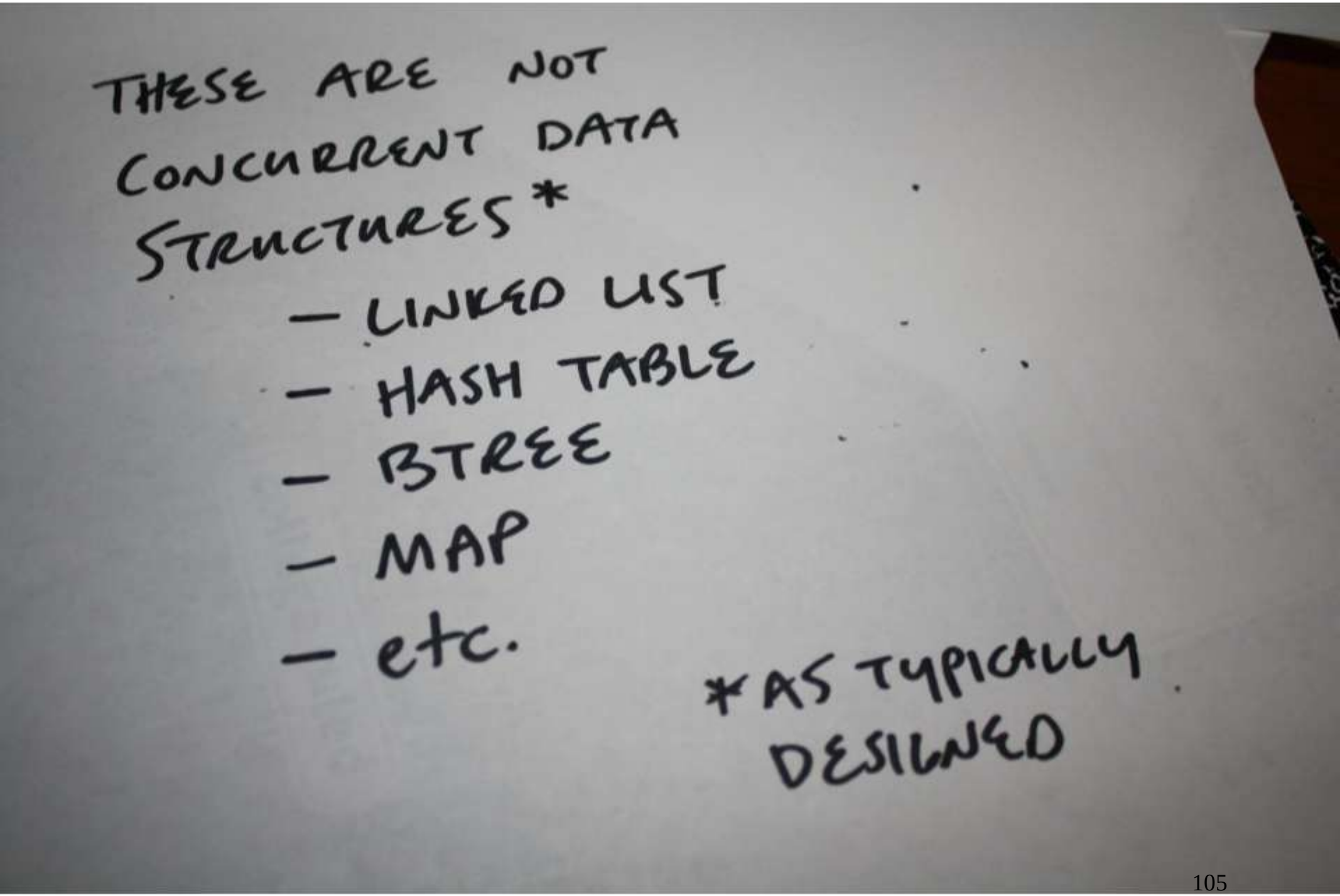

SOLVE EACH SEPERATELY
TEXTURES
POS DATA
STATE DATA
DRAW
POS DATA'
UPDATE GHOSTS
STATE DATA'


POS DATA
STATE DATA
ONLY LOAD WHAT YOU NEED!
DATA'
UPDATE GHOSTS
STATE DATA'


INSOMNIAC
EXAMPLE:
Aligned Box
and
Friends...

TYPICAL PROBLEM

BLINDLY APPLY ALGORITHM TO CONCURRENT H/W


THESE ARE NOT
CONCURRENT DATA
STRUCTURES*
– LINKED LIST
– HASH TABLE
– BTREE
– MAP
– etc.
*AS TYPICALLY
DESIGNED

ISSUES:

- NOT UNDERSTANDING CONCURRENCY CHARACTERISTICS OF THE PLATFORM

- NOT UNDERSTANDING OR DEFINING THE REAL CONSTRAINTS OF THE PROBLEM.

TYPICAL PROBLEM

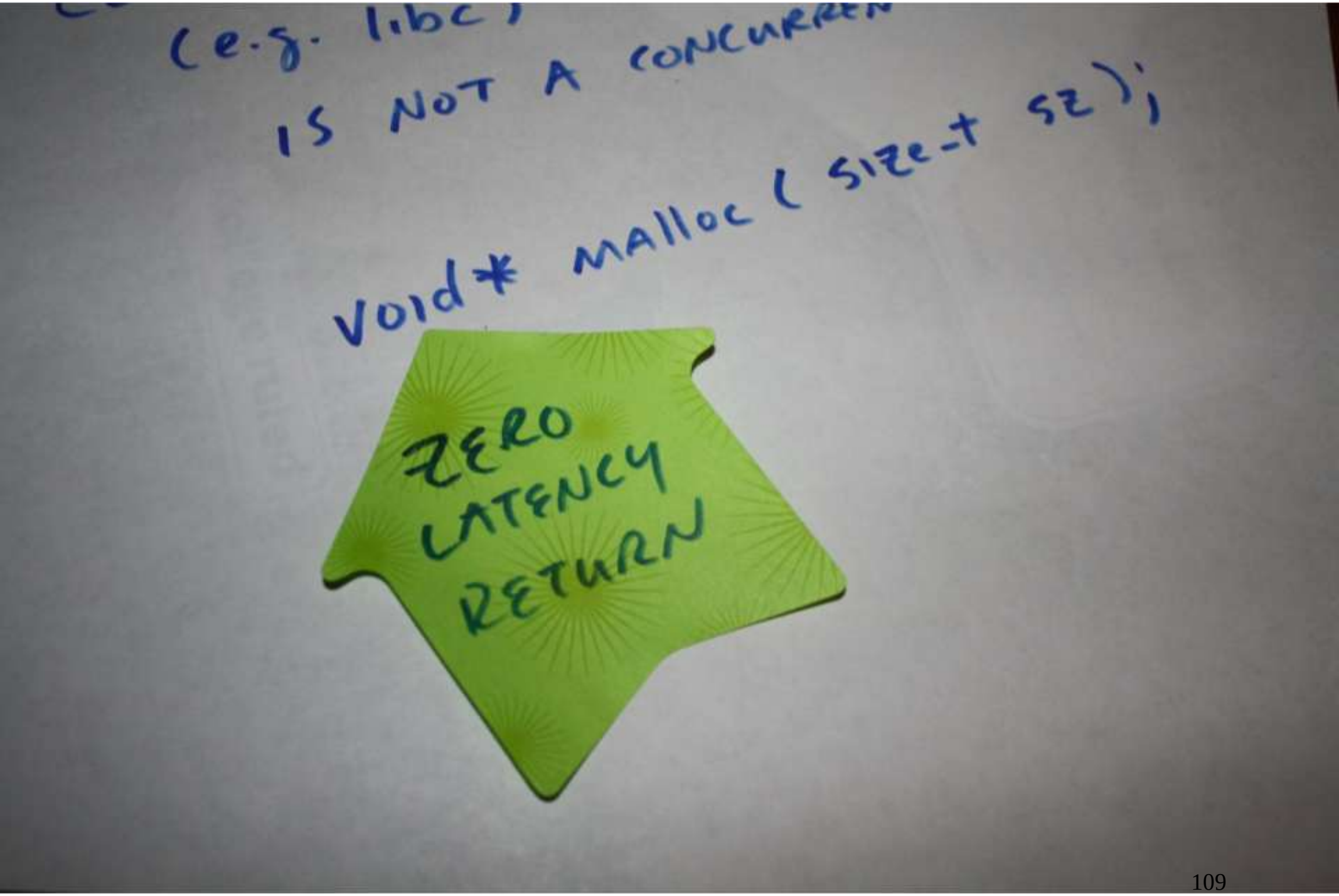
RETURN VALUES

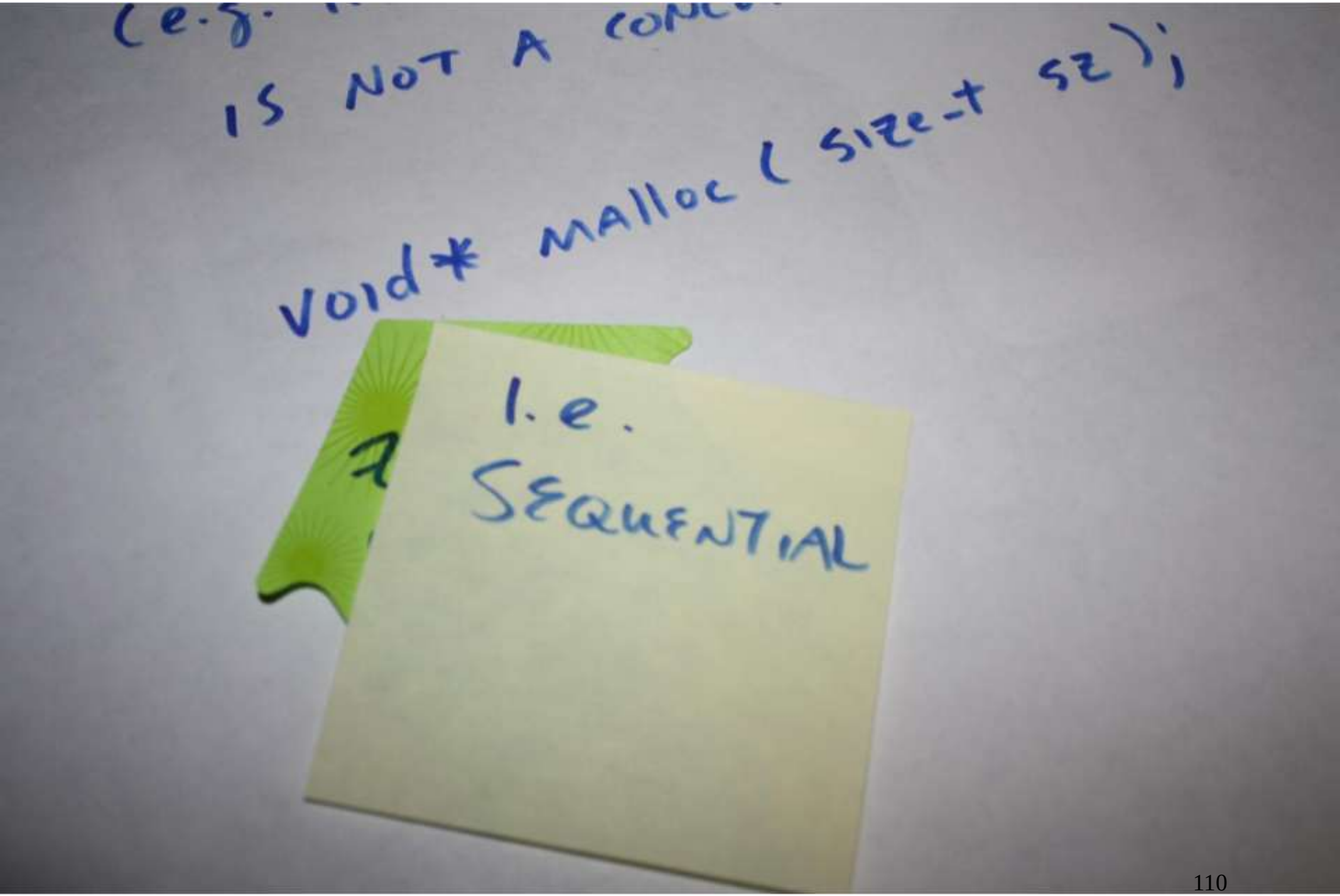
I.E. NOT SOLVING FOR LATENCY

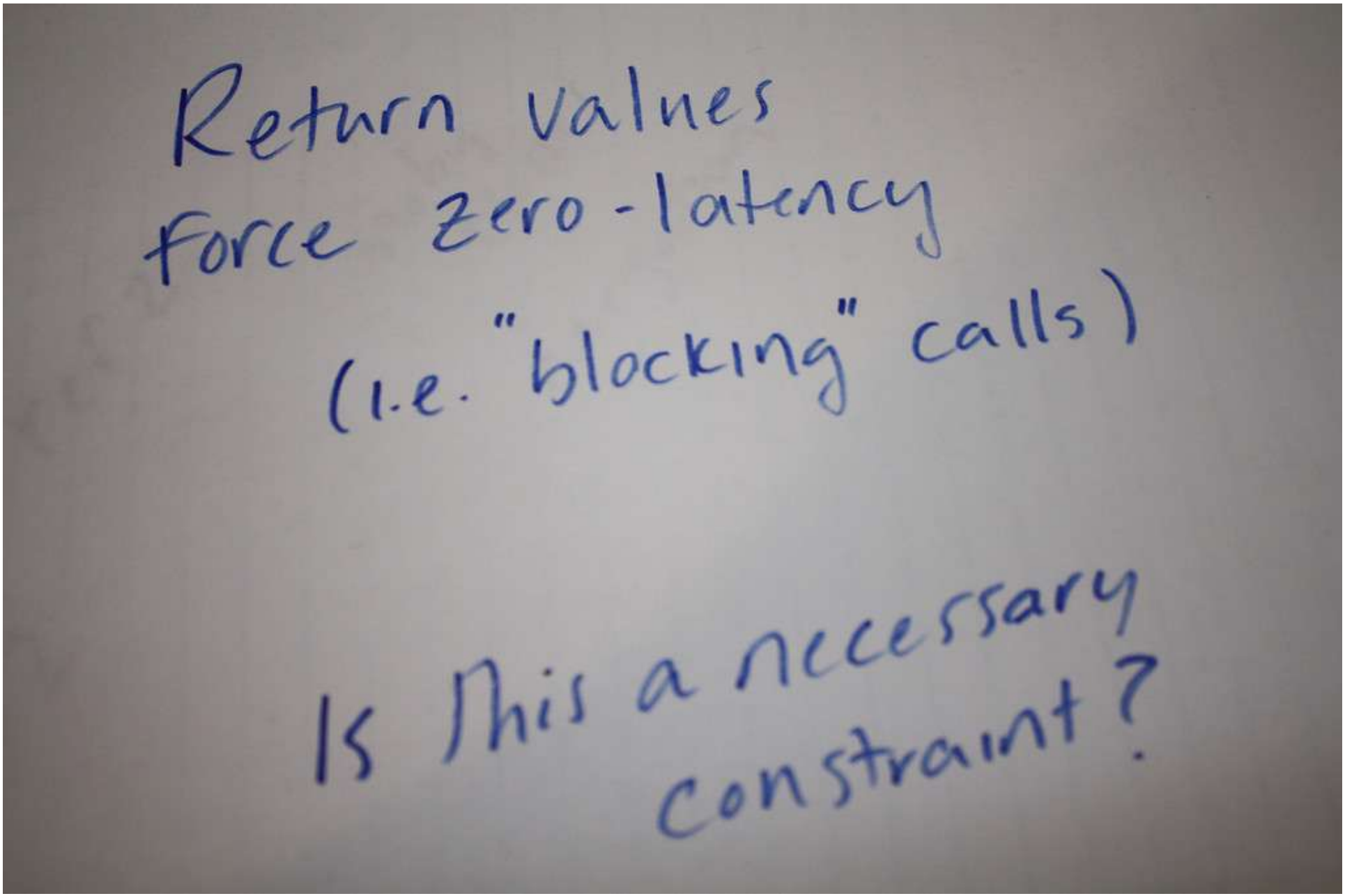
CLASSIC API DESIGN
(e.g. libc)

IS NOT A CONCURRENT DESIGN

void* malloc (size_t sz);


(e.g. libc)
IS NOT A CONCURREN
void* malloc (size_t sz);
ZERO
LATENCY
RETURN


(e.g.
IS NOT A
void* malloc (size_t sz);
I.e.
SEQUENTIAL


Return values
force zero-latency
(i.e. "blocking" calls)
Is this a necessary
constraint?

e.g. implicit returns

setContext()

setPosition()

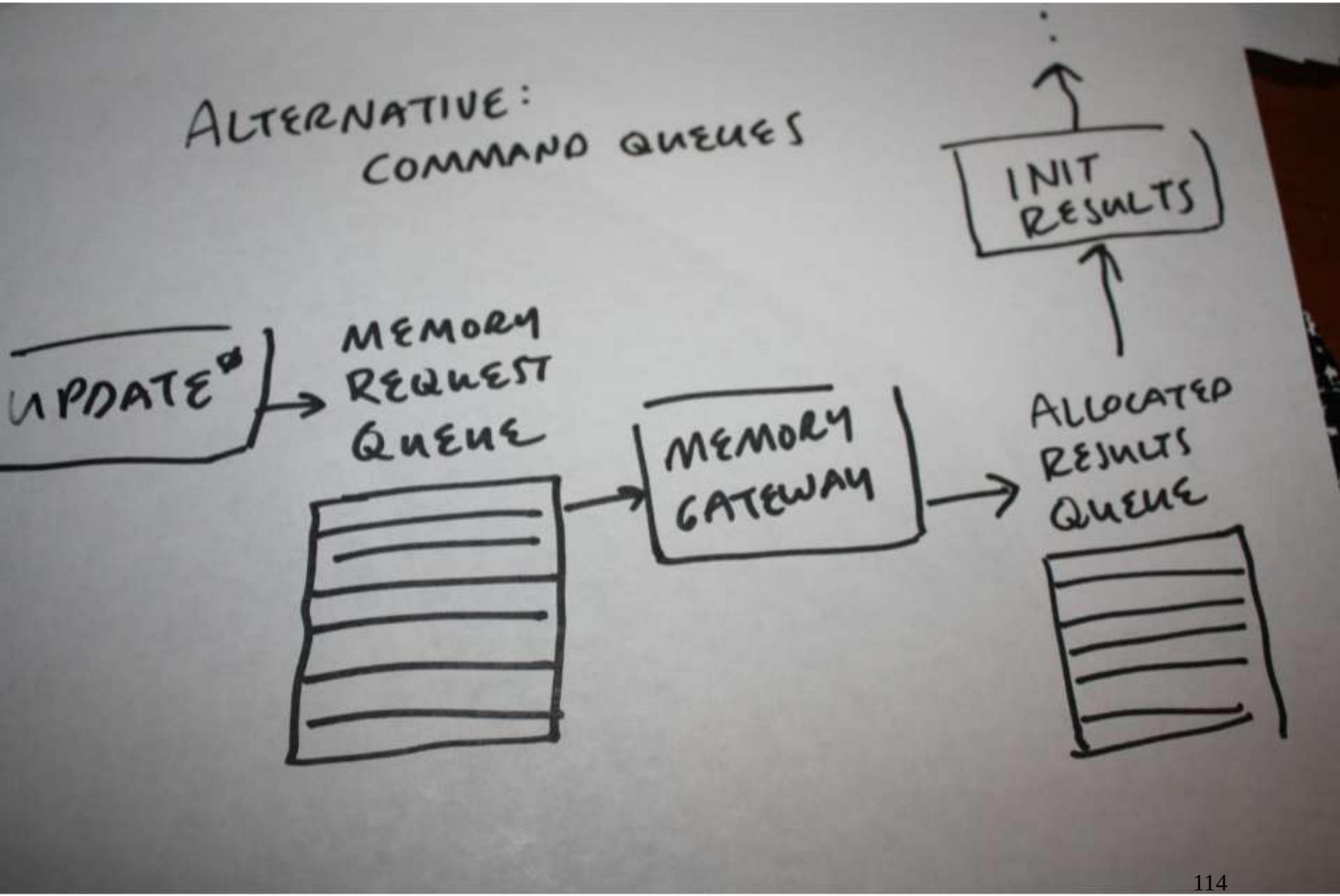
updateState()

e.g. explicit returns

malloc()

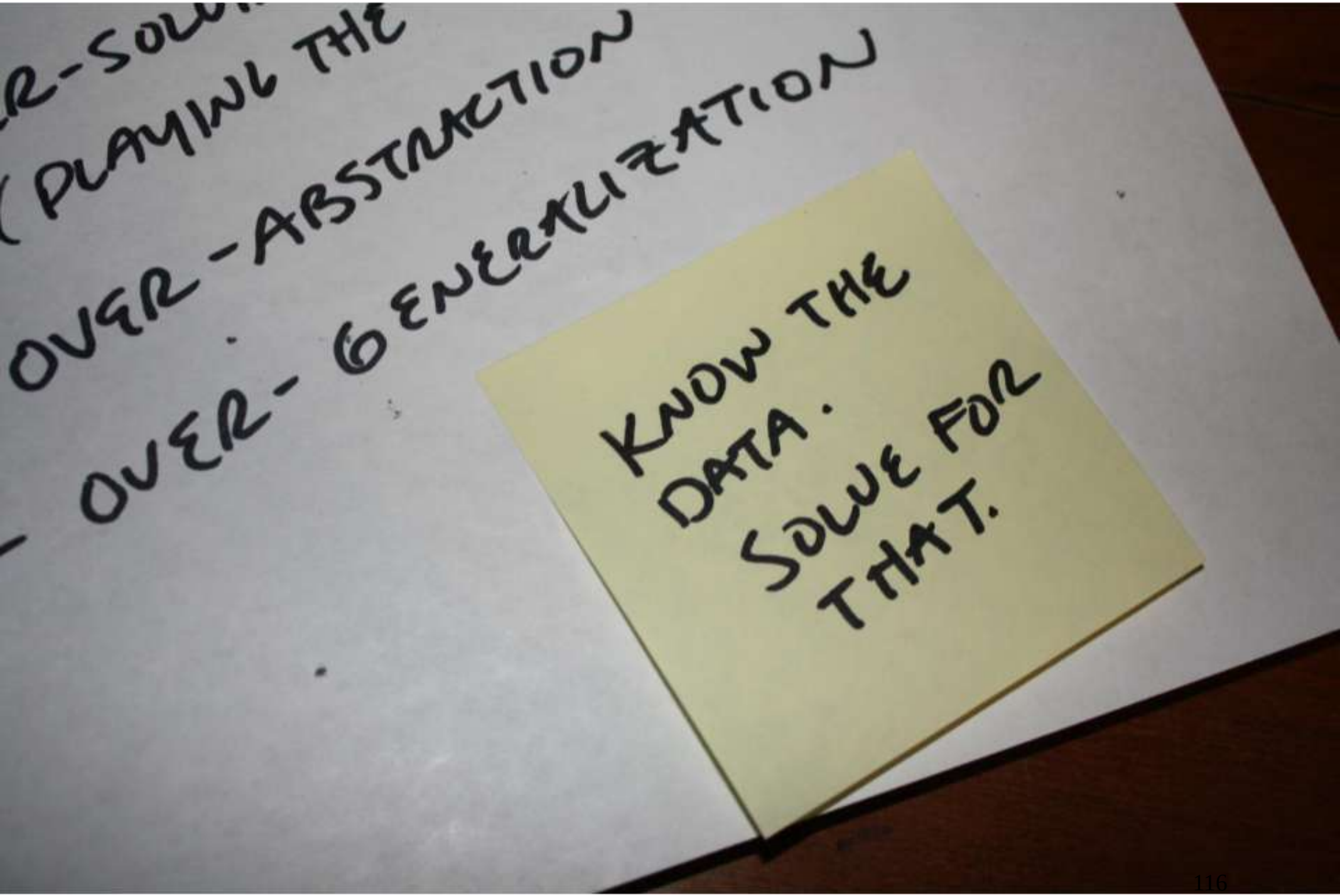
fread()

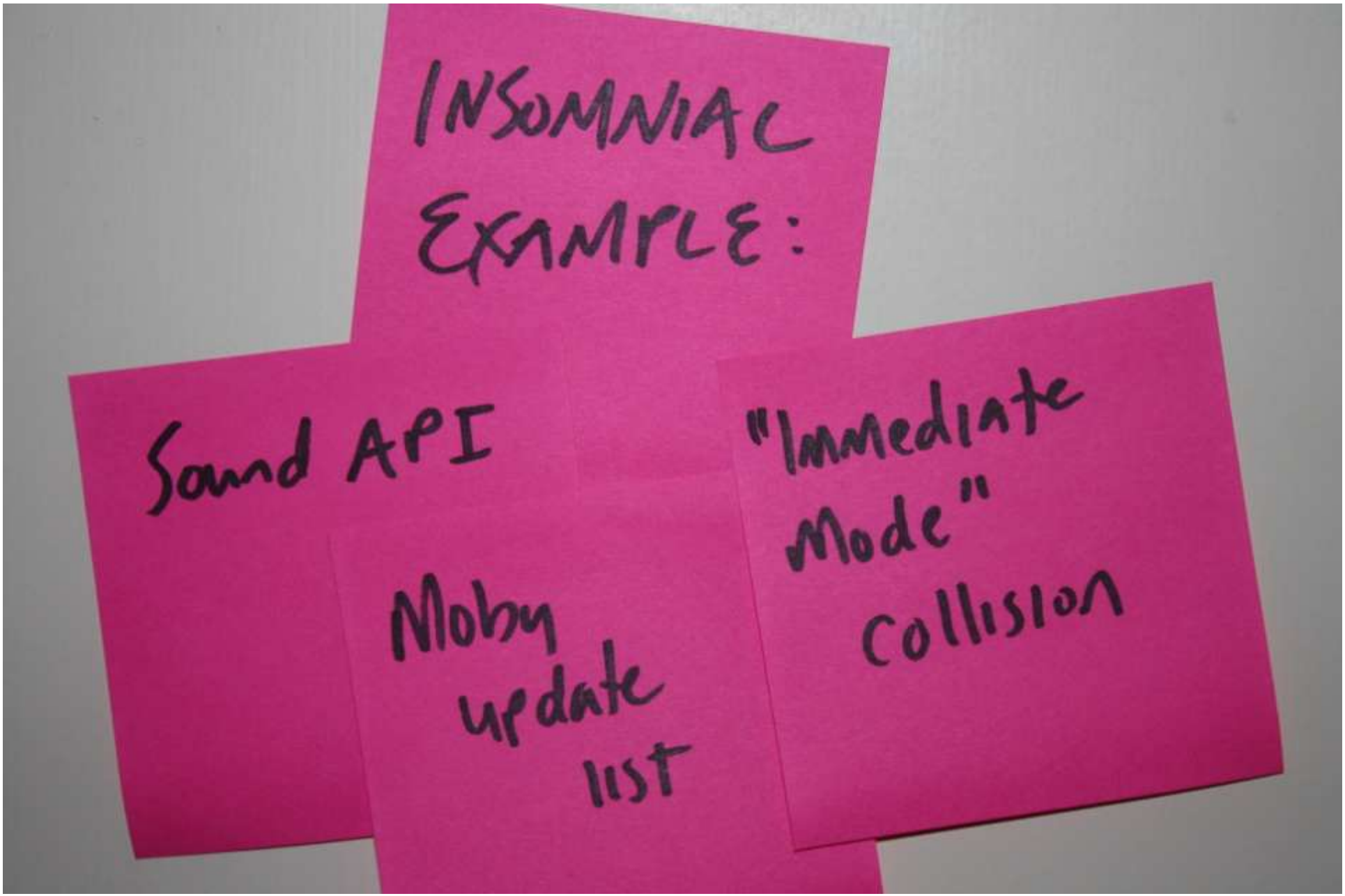
getRotation()

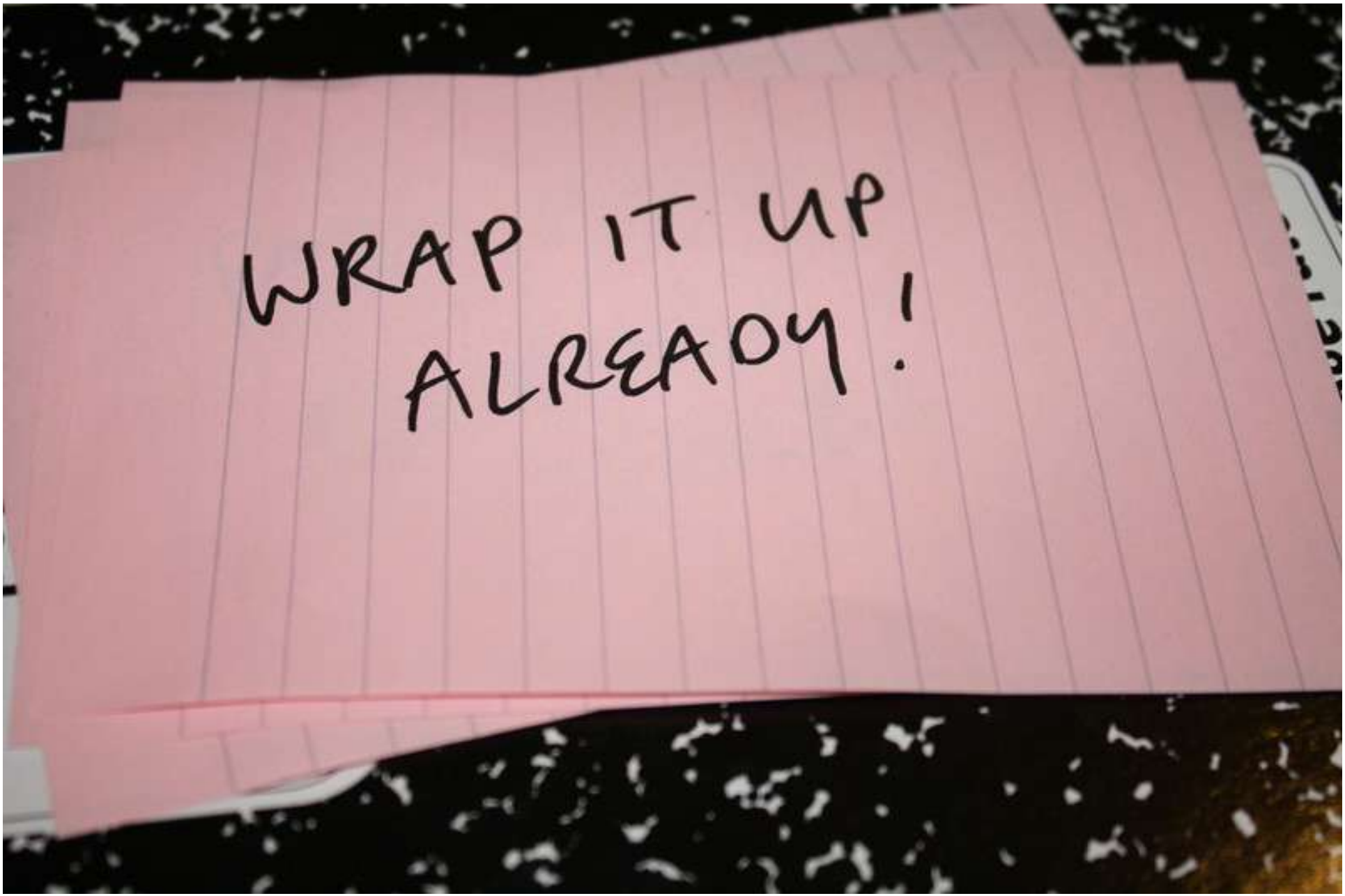

ALTERNATIVE:
COMMAND QUEUES
INIT RESULTS
UPDATE*
MEMORY REQUEST QUEUE
MEMORY GATEWAY
ALLOCATED RESULTS QUEUE

OTHER TYPICAL PROBLEMS

- OVER-SOLVING PROBLEMS
  (PLAYING THE "WHAT IF" GAME)
- OVER-ABSTRACTION
- OVER-GENERALIZATION


R-SOLU
(PLAYING THE
OVER-ABSTRACTION
OVER-GENERALIZATION
OVER
KNOW THE
DATA.
SOLVE FOR
THAT.


INSOMNIAC
EXAMPLE:
Sound API
Moby
update
list
"Immediate
Mode"
Collision


WRAP IT UP
ALREADY!

TRUTHS

1. HARDWARE IS THE PLATFORM
2. DESIGN AROUND THE DATA, NOT AN IDEALIZED WORLD
3. YOUR MAIN RESPONSIBILITY IS TO TRANSFORM DATA, SOLVE THAT FIRST, NOT THE CODE DESIGN.


THE END!