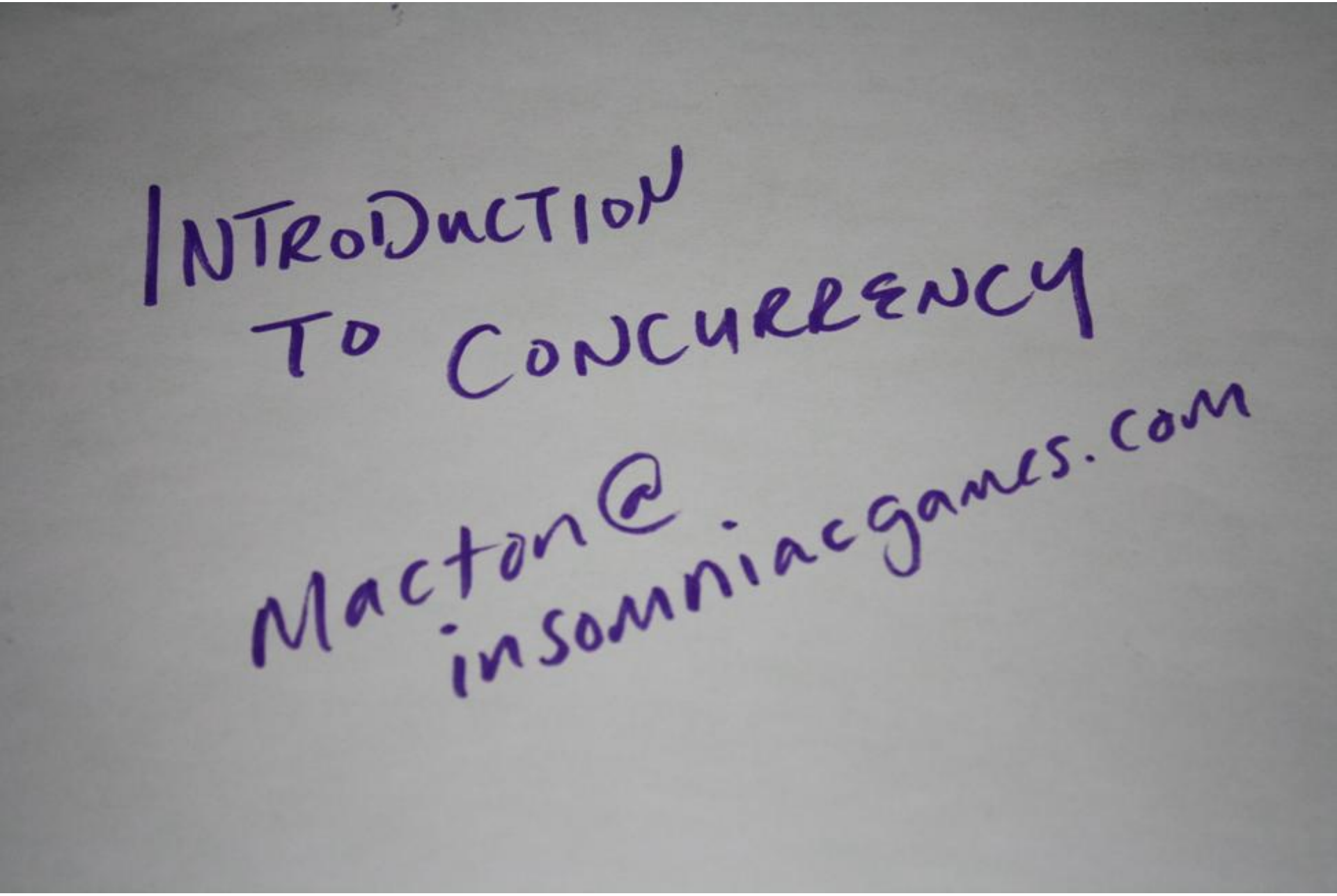
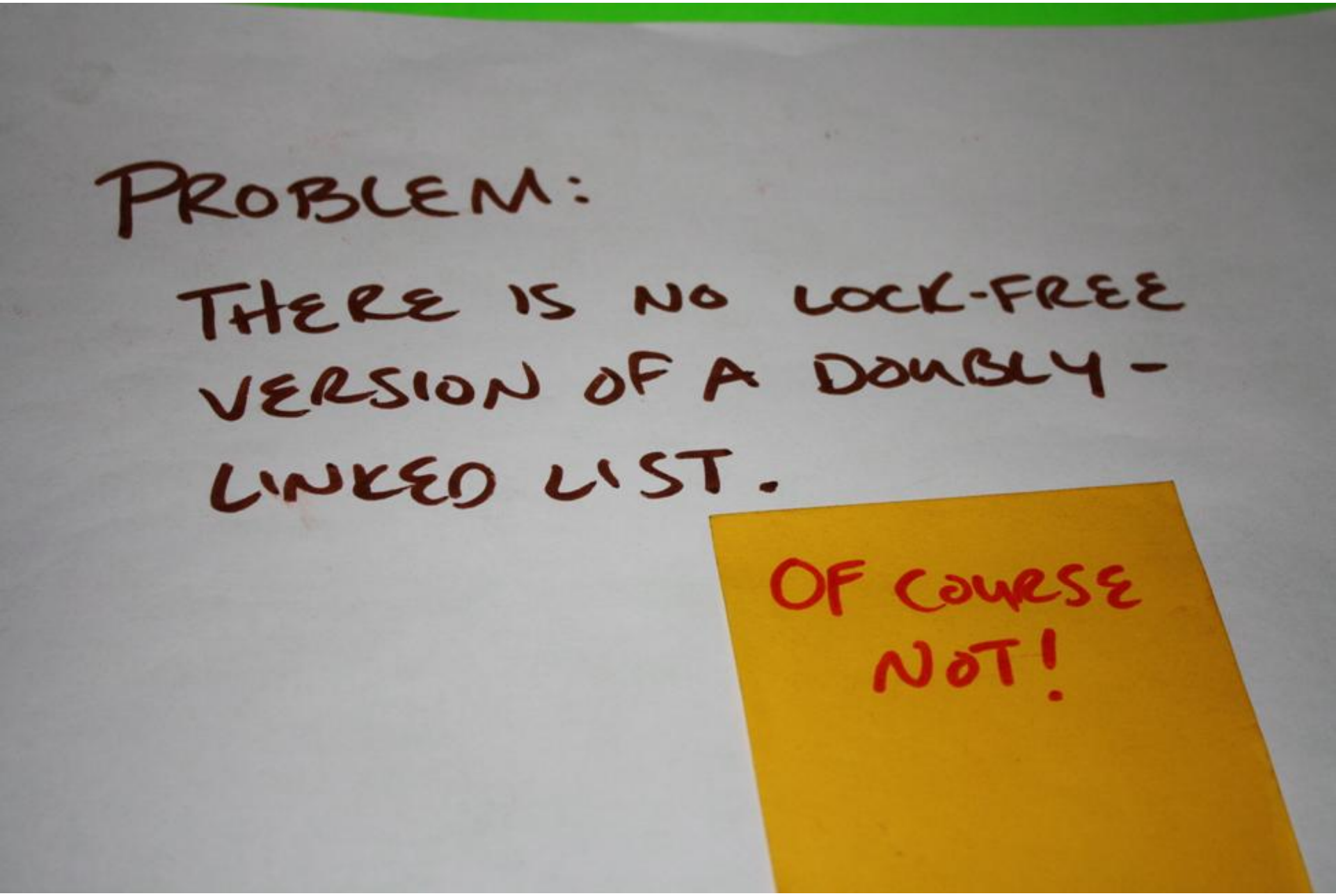
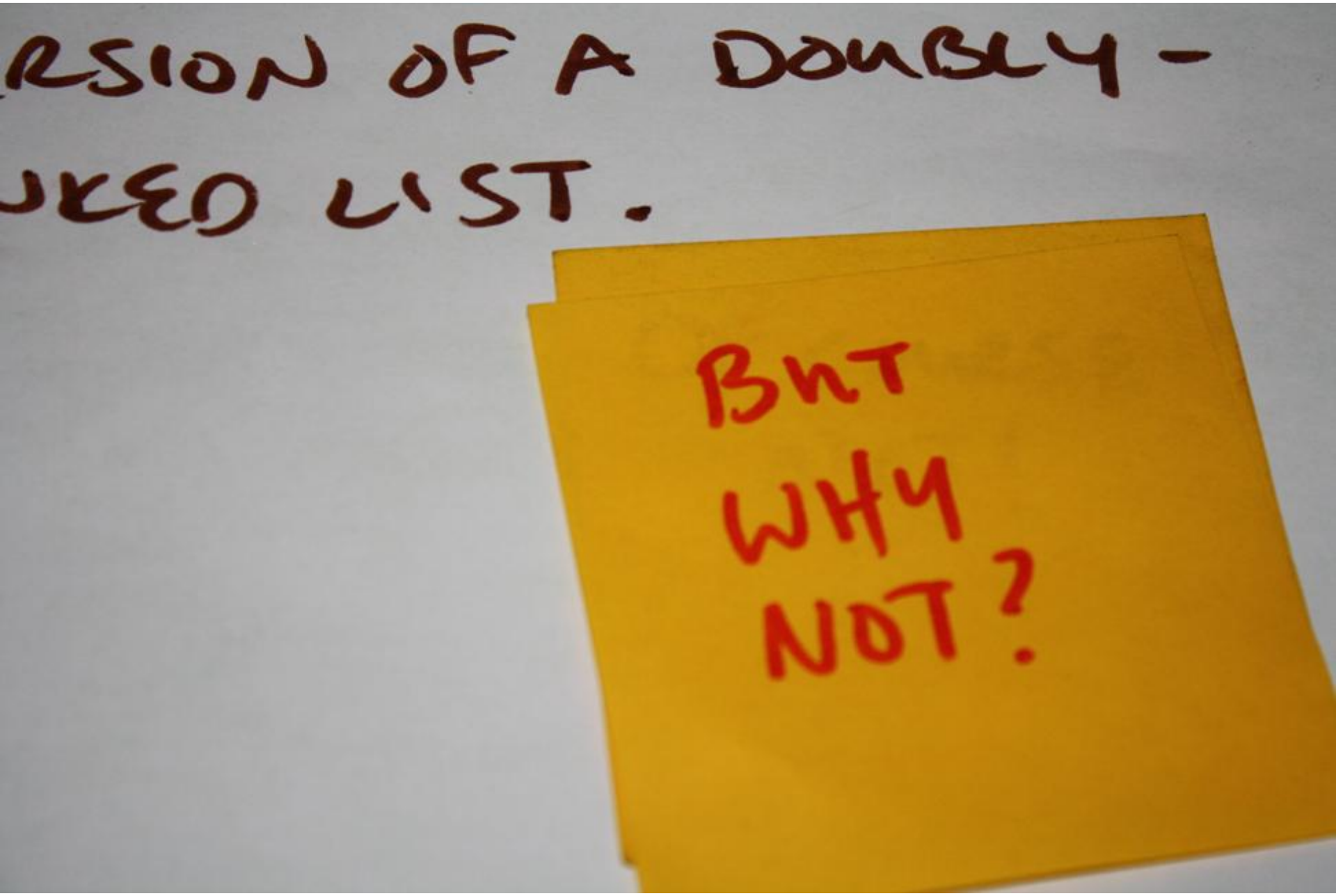

Introduction
To Concurrency
Macton@
insomniacgames.com

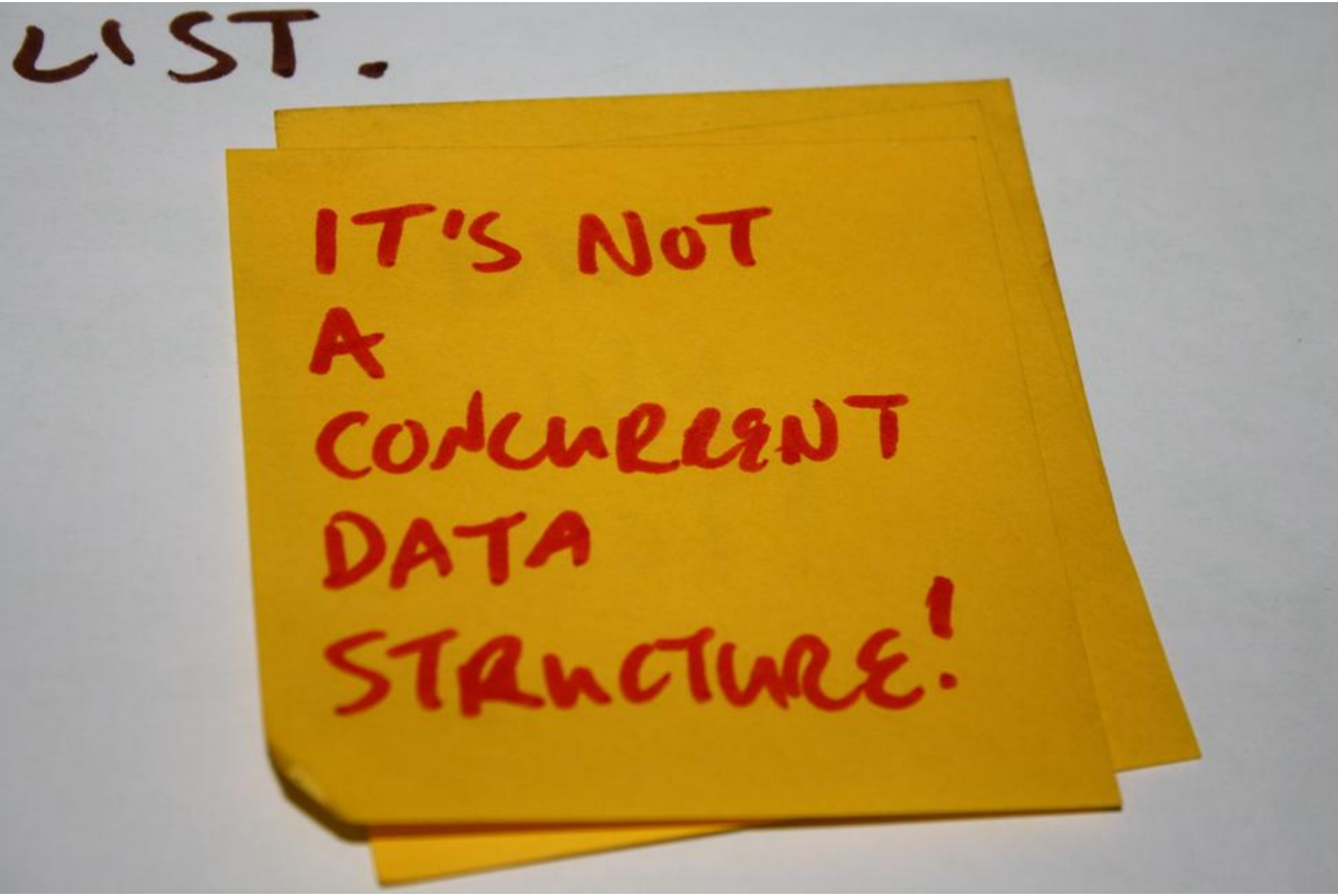
Let's start w/ a well-known "problem" and work backward...

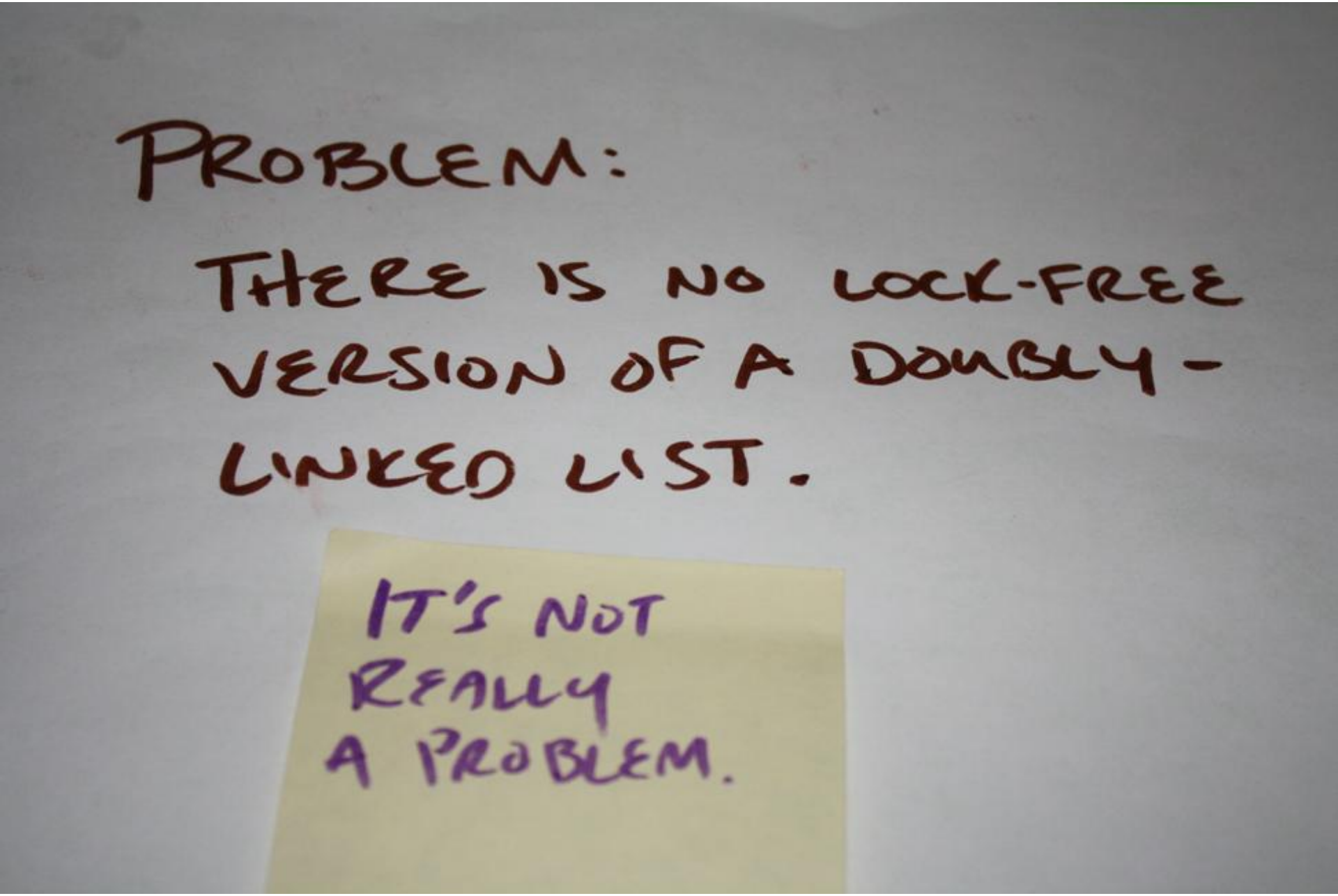
PROBLEM:

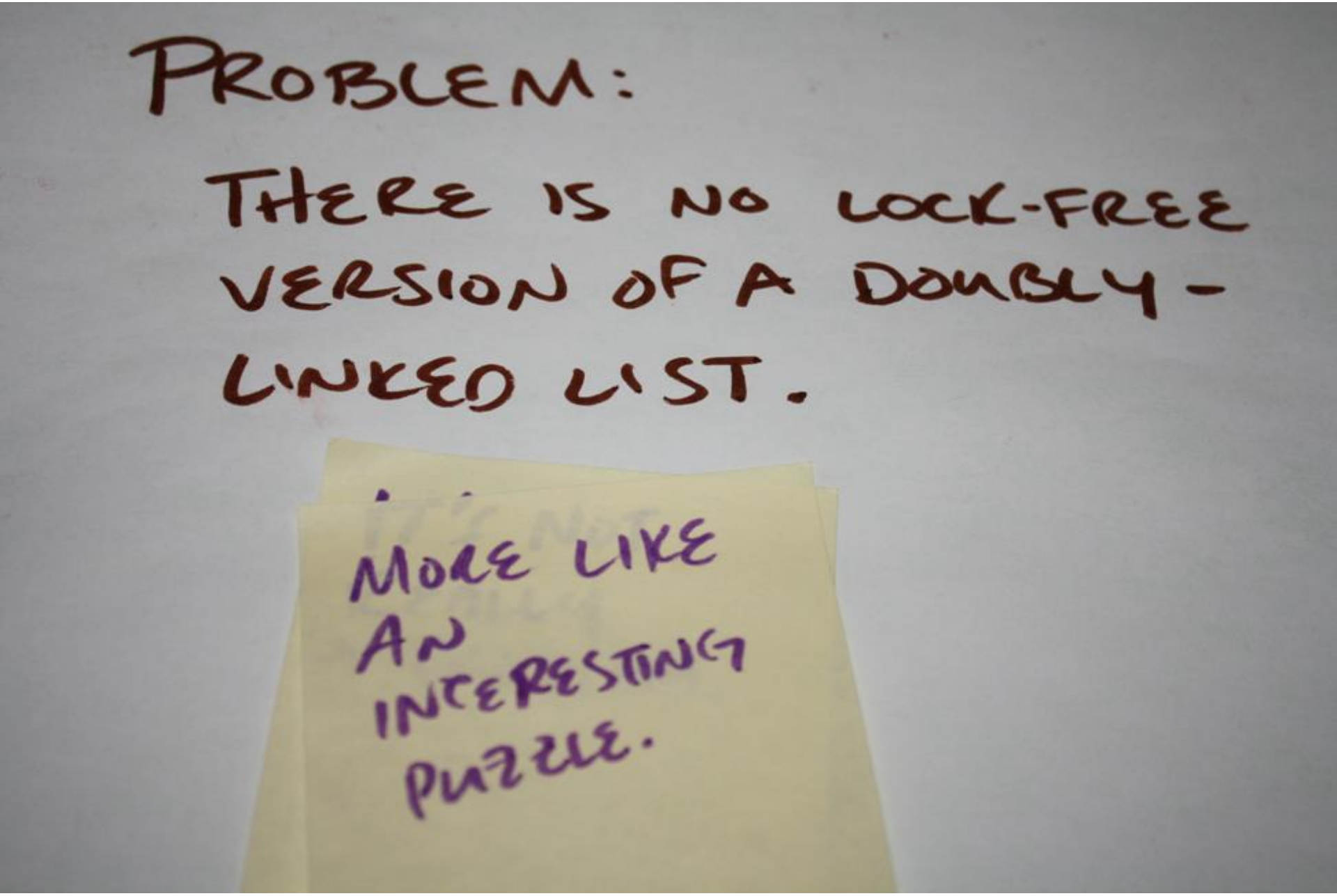
THERE IS NO LOCK-FREE VERSION OF A DOUBLY-LINKED LIST.


PROBLEM:
THERE IS NO LOCK-FREE VERSION OF A DOUBLY-LINKED LIST.
OF COURSE NOT!


RSION OF A DOUBLY-
UKED LIST.
BUT
WHY
NOT?


LIST.
IT'S NOT
A
CONCURRENT
DATA
STRUCTURE!


PROBLEM:
THERE IS NO LOCK-FREE VERSION OF A DOUBLY-LINKED LIST.
IT'S NOT REALLY A PROBLEM.

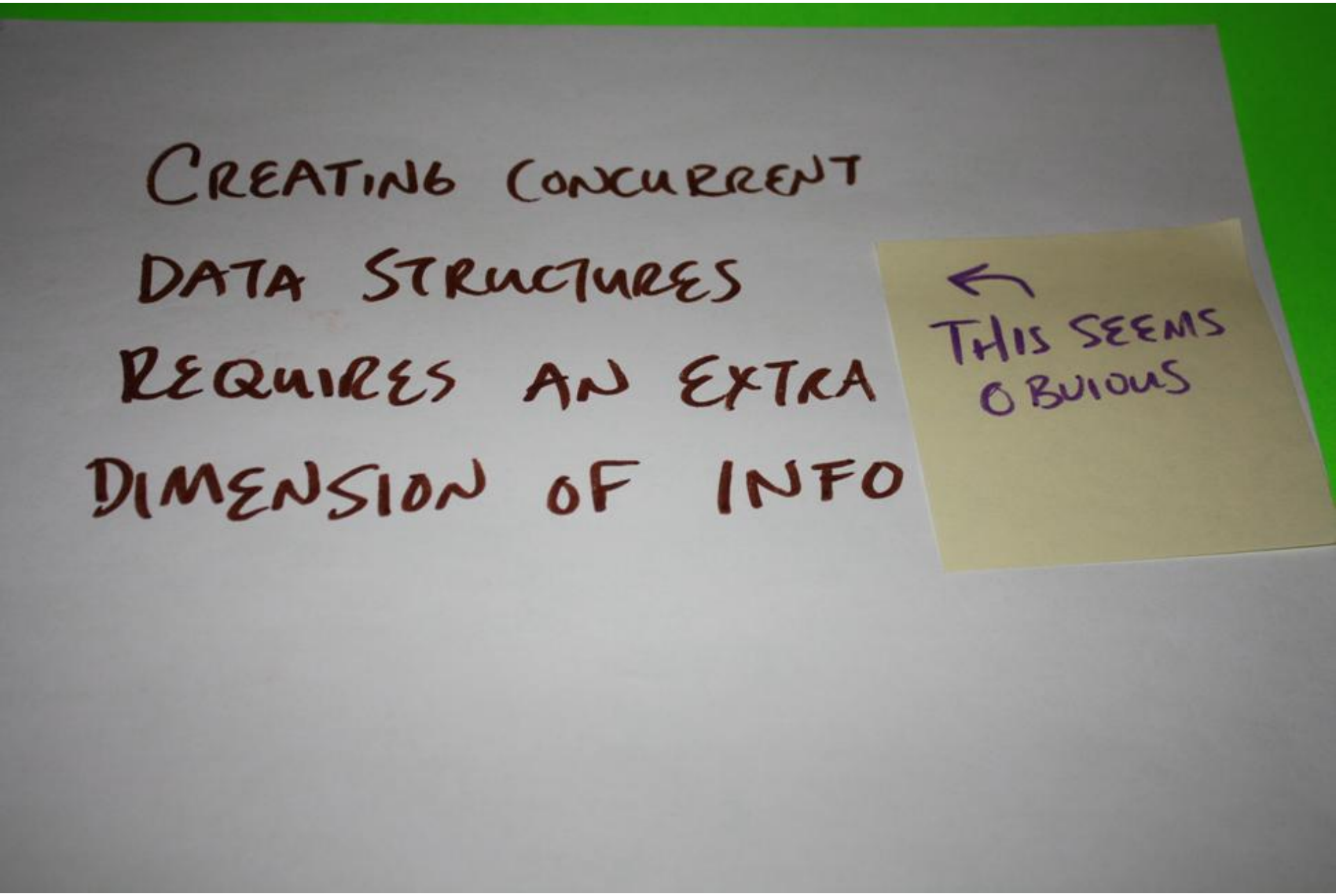

PROBLEM:
THERE IS NO LOCK-FREE VERSION OF A DOUBLY-LINKED LIST.
MORE LIKE AN INTERESTING PUZZLE.

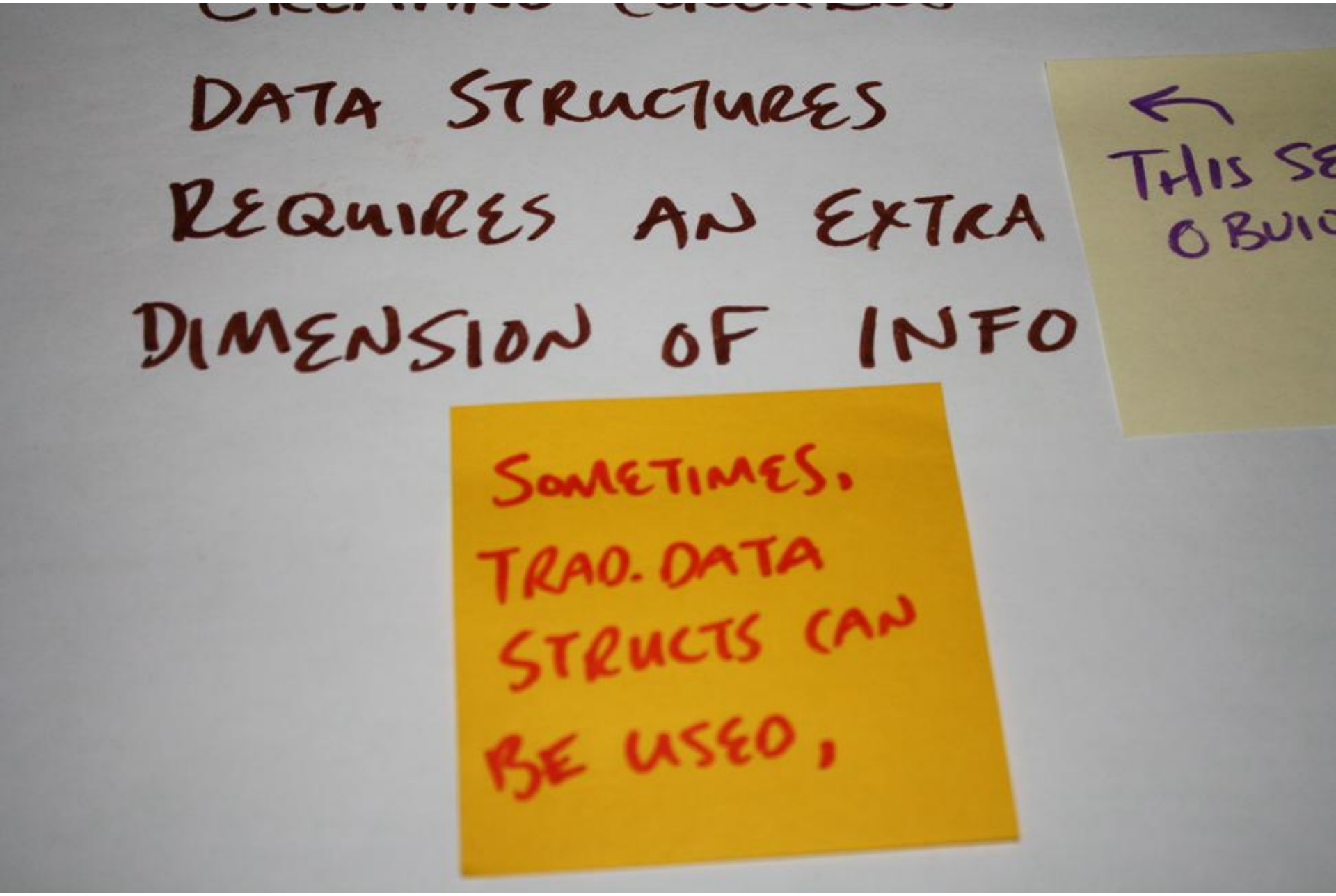
PROBLEM:

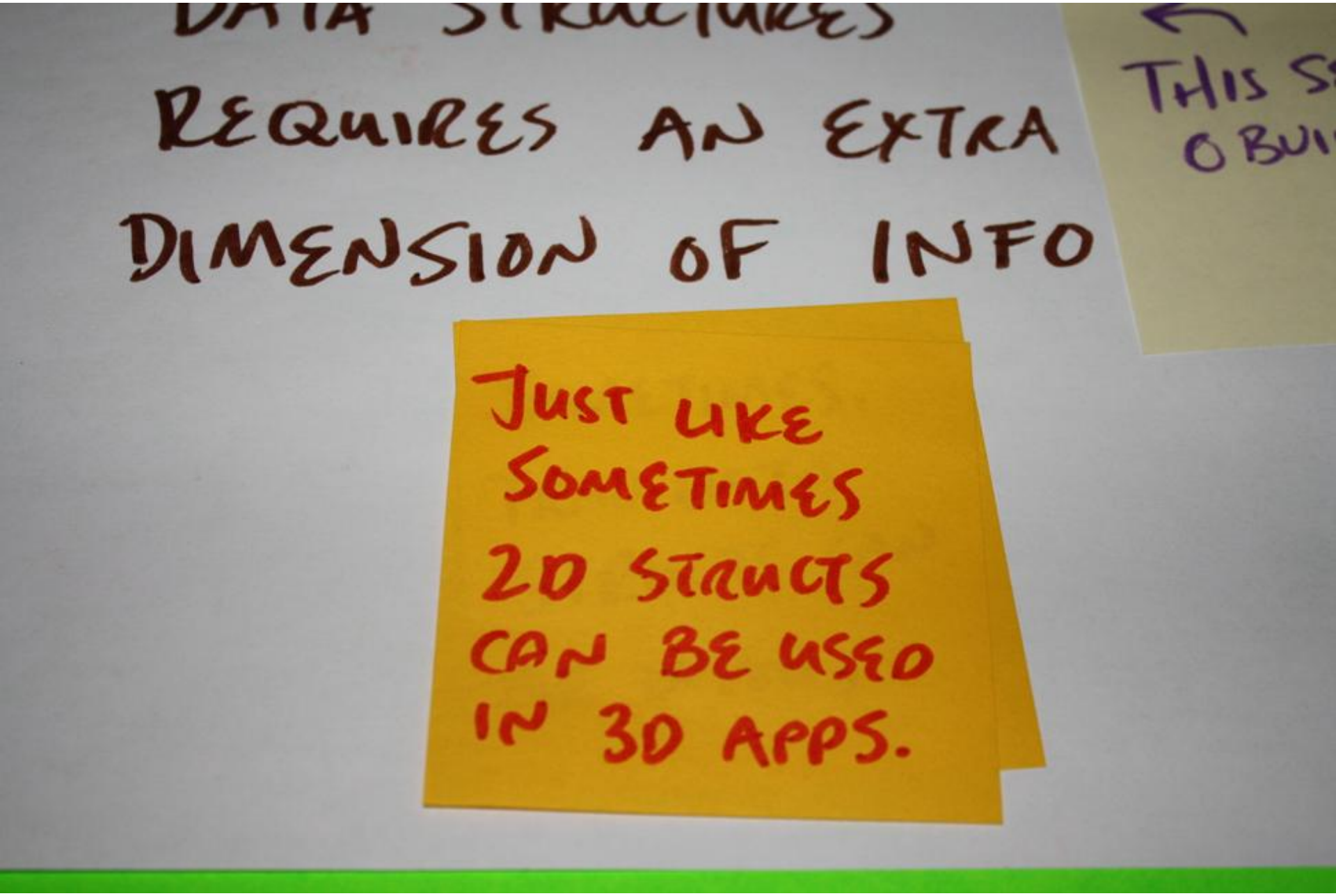
THERE IS NO LOCK-FREE VERSION OF A DOUBLY-LINKED LIST.

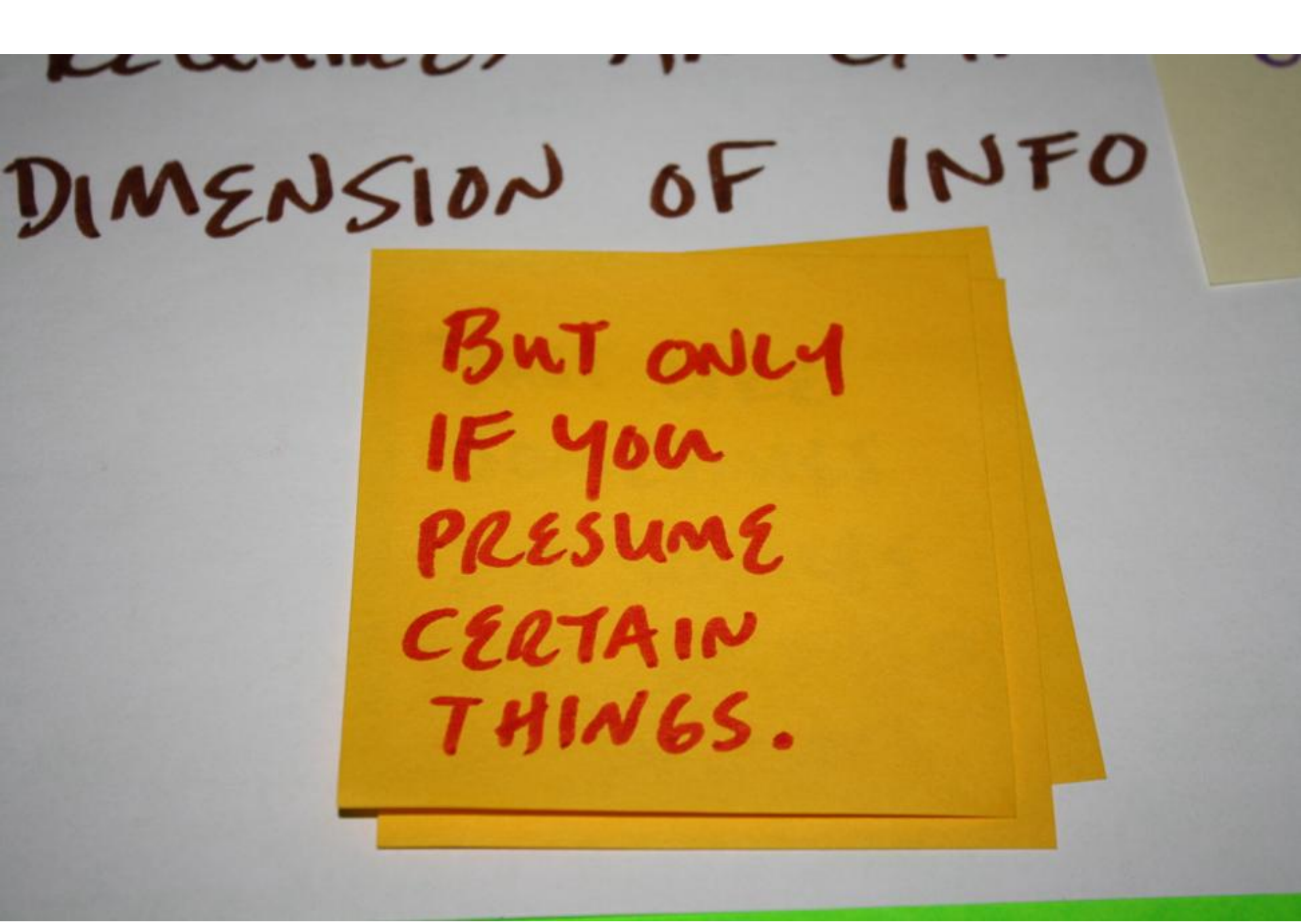
IT'S THE WRONG LEVEL OF ABSTRACTION FOR CONCURRENCY.

CREATING CONCURRENT DATA STRUCTURES REQUIRES AN EXTRA DIMENSION OF INFO

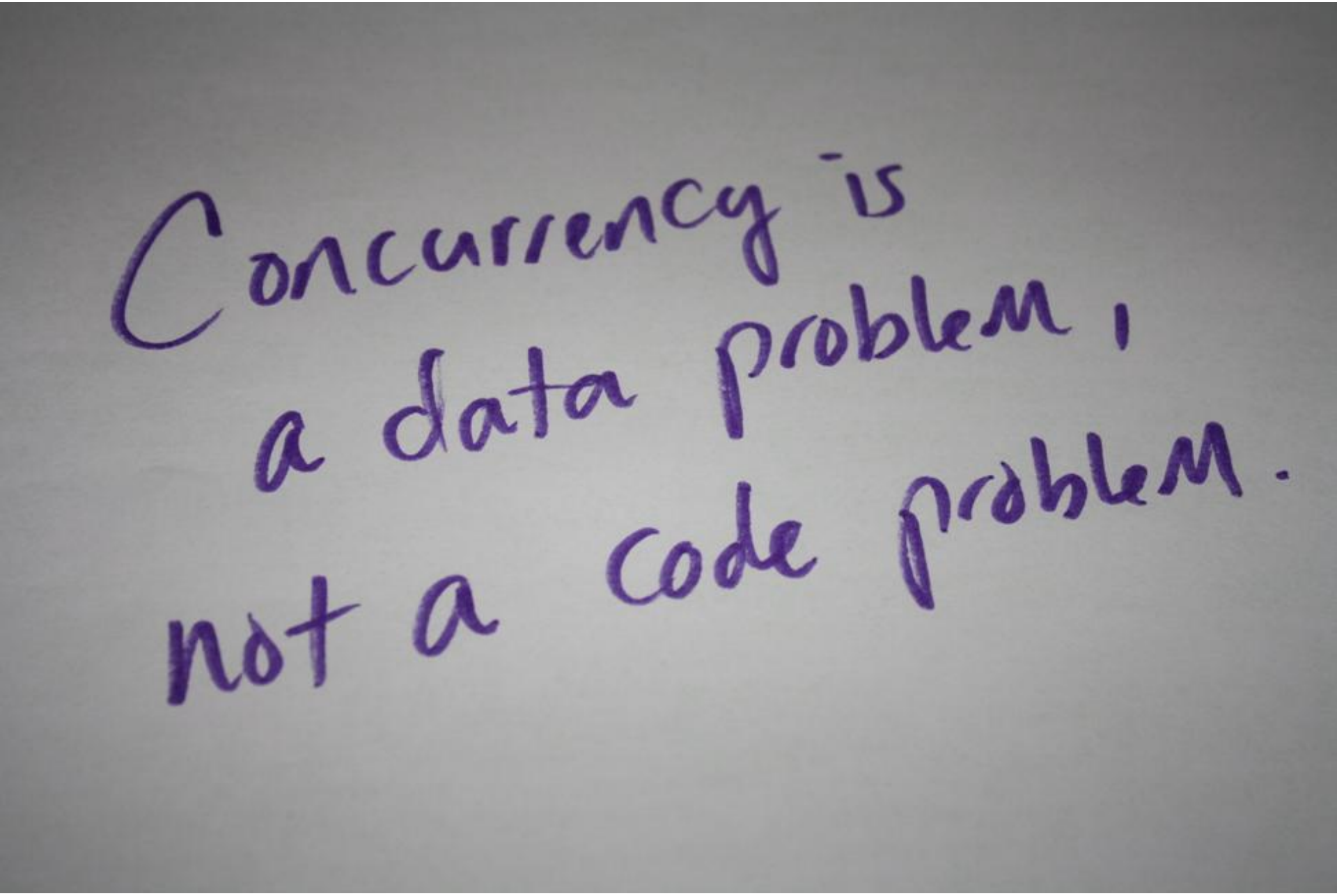

CREATING CONCURRENT
DATA STRUCTURES
REQUIRES AN EXTRA
DIMENSION OF INFO
THIS SEEMS
OBVIOUS

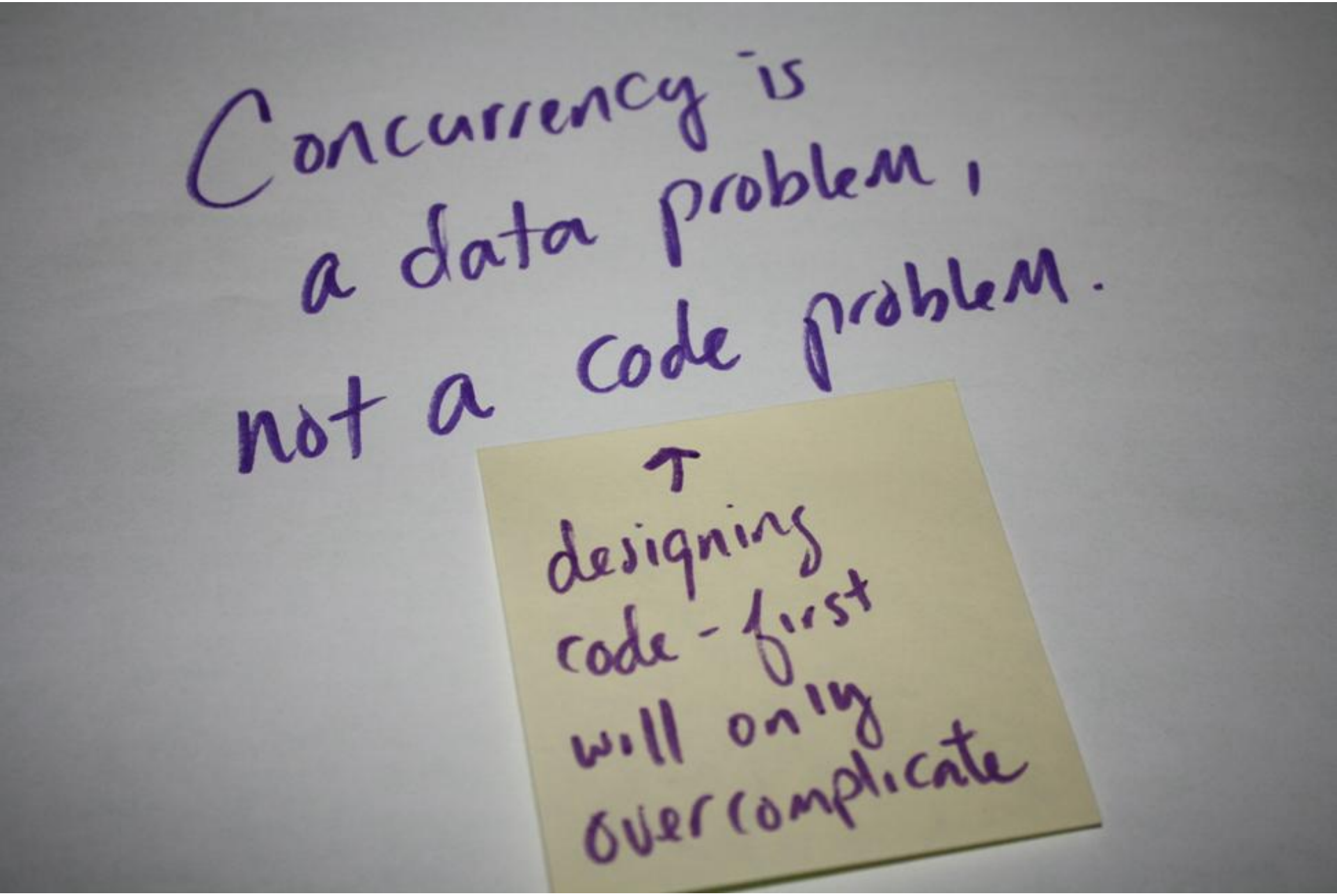

DATA STRUCTURES
REQUIRES AN EXTRA
DIMENSION OF INFO
SOMETIMES,
TRAD. DATA
STRUCTS CAN
BE USED,


DATA STRUCTURES
REQUIRES AN EXTRA
DIMENSION OF INFO
JUST LIKE
SOMETIMES
2D STRUCTS
CAN BE USED
IN 3D APPS.

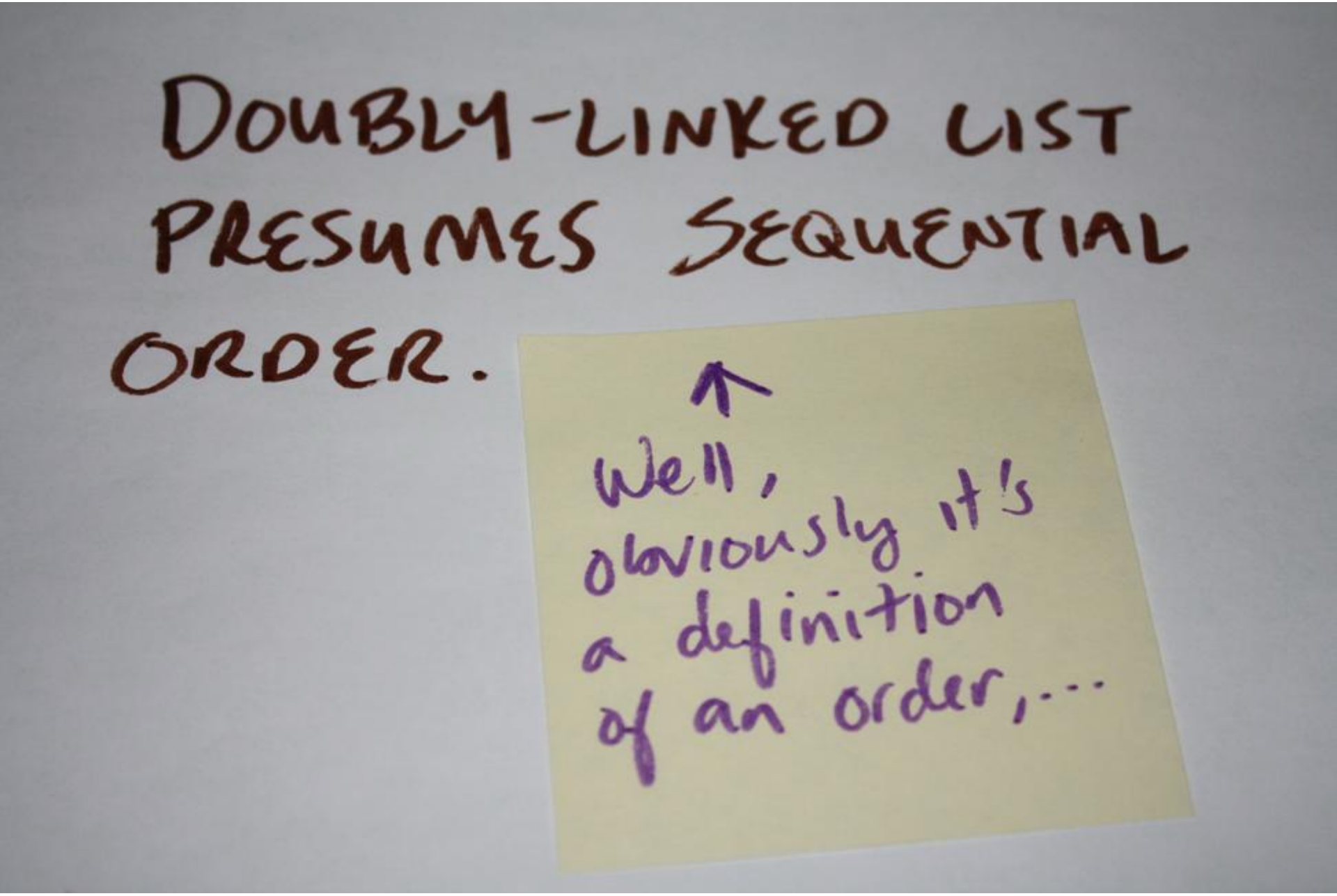

DIMENSION OF INFO
BUT ONLY
IF YOU
PRESUME
CERTAIN
THINGS.


DIMENSION
Remember -
It's Always
About The
data!
But
IF yo
PRESU
CERTA
THIN


Concurrency is
a data problem,
not a code problem.


Concurrency is
a data problem,
not a code problem.
↑
designing
code-first
will only
overcomplicate

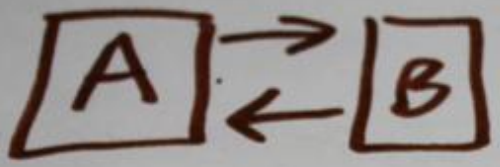
DOUBLY-LINKED LIST PRESUMES SEQUENTIAL ORDER.


DOUBLY-LINKED LIST PRESUMES SEQUENTIAL ORDER.
↑
Well, obviously it's a definition of an order, ...

ESUMES SEQUENTIAL

DER.

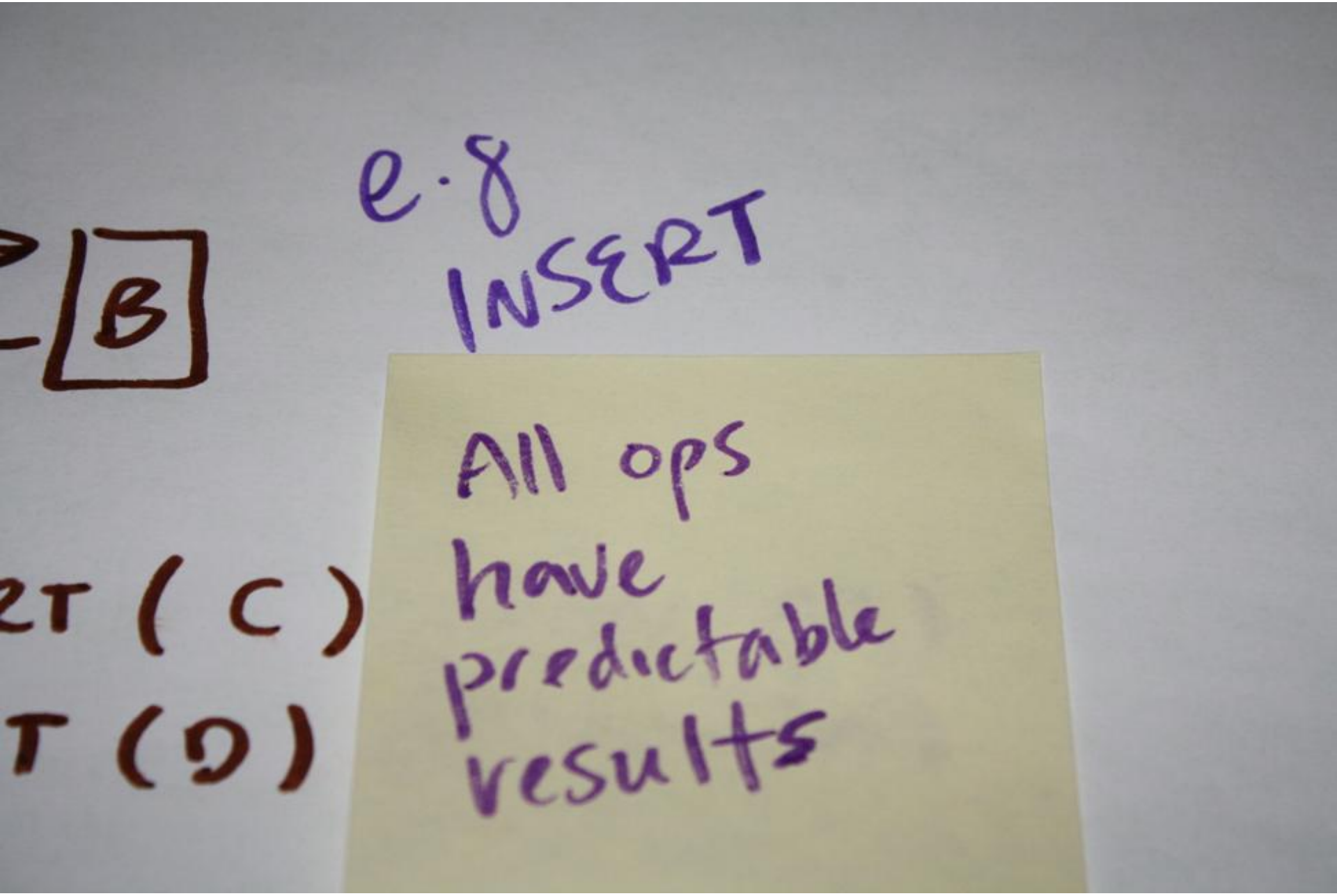
But xforms of the data are also implicitly ordered.



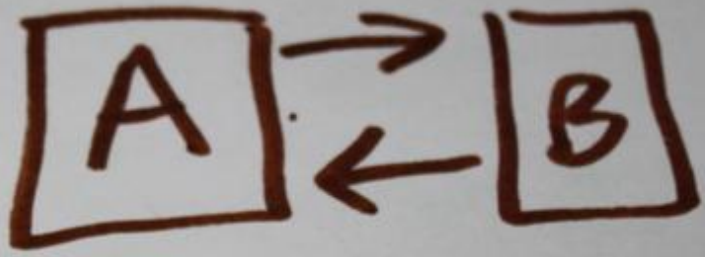
e.g
INSERT

- INSERT (C) after (A)
- INSERT (D) after (A)

HAS GUARANTEED RESULT:


B
e.g
INSERT
All ops
have
predictable
results

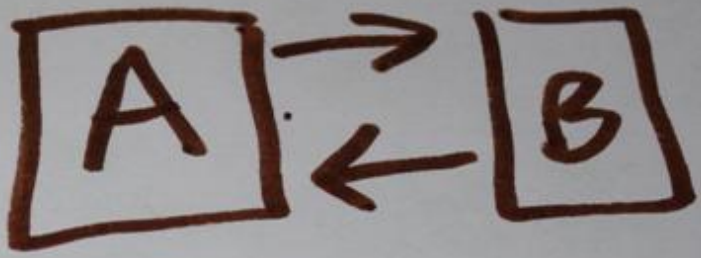
the data struct only exists to facilitate the results.



e

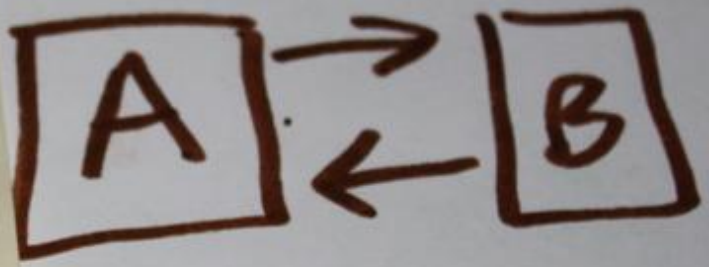
- INSERT (C
- INSERT (D

What Should The results be?

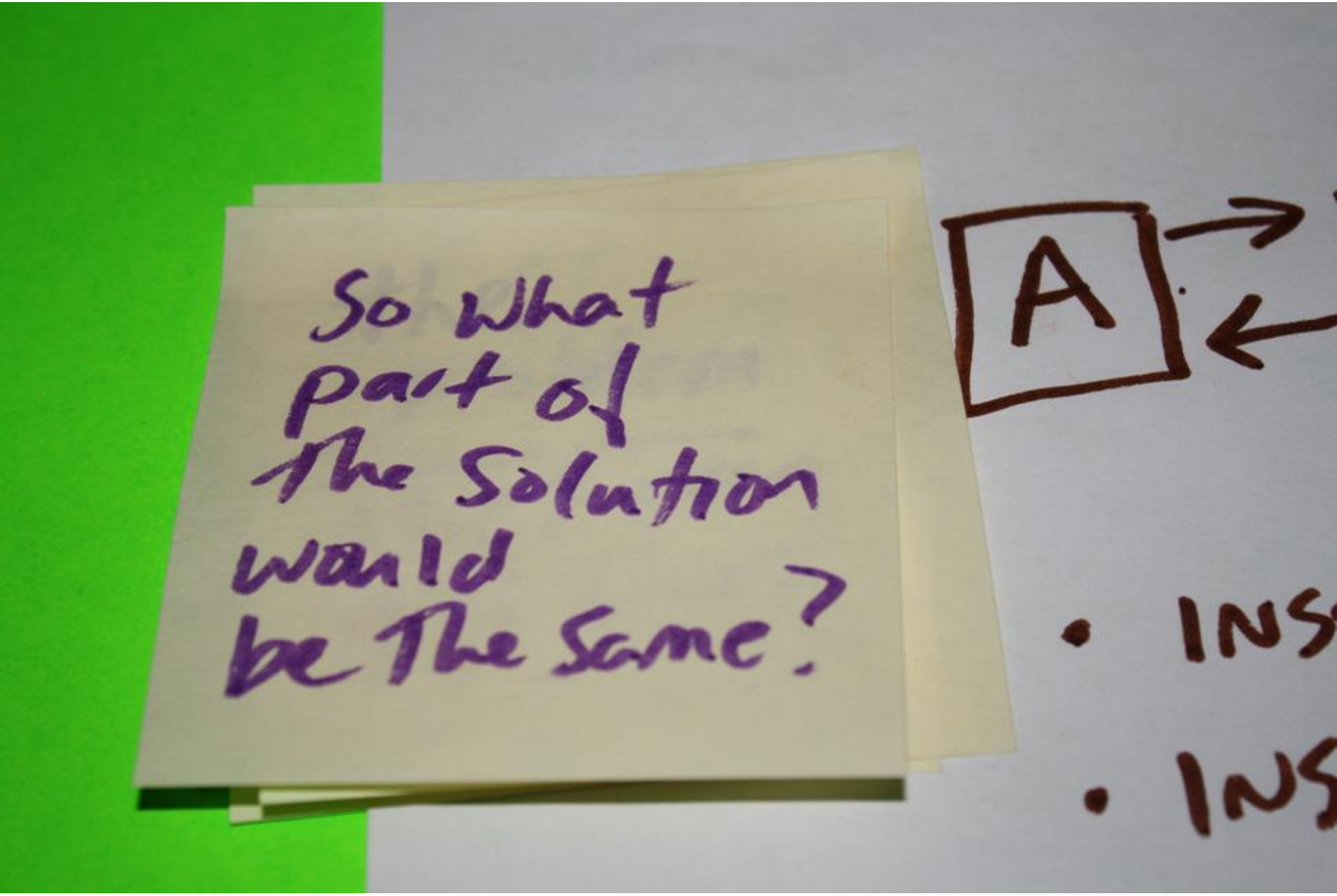


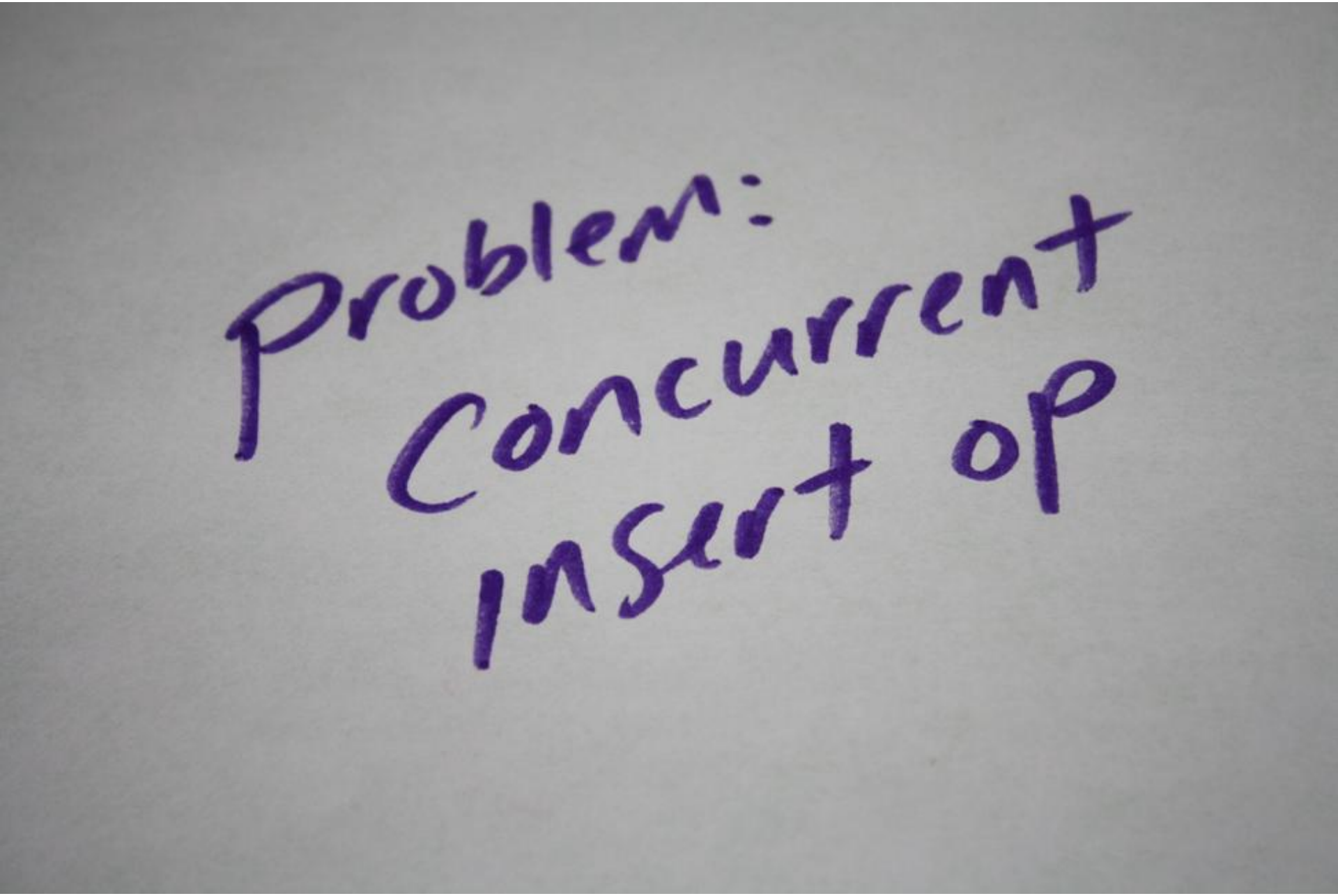
- INSERT (
- INSERT (

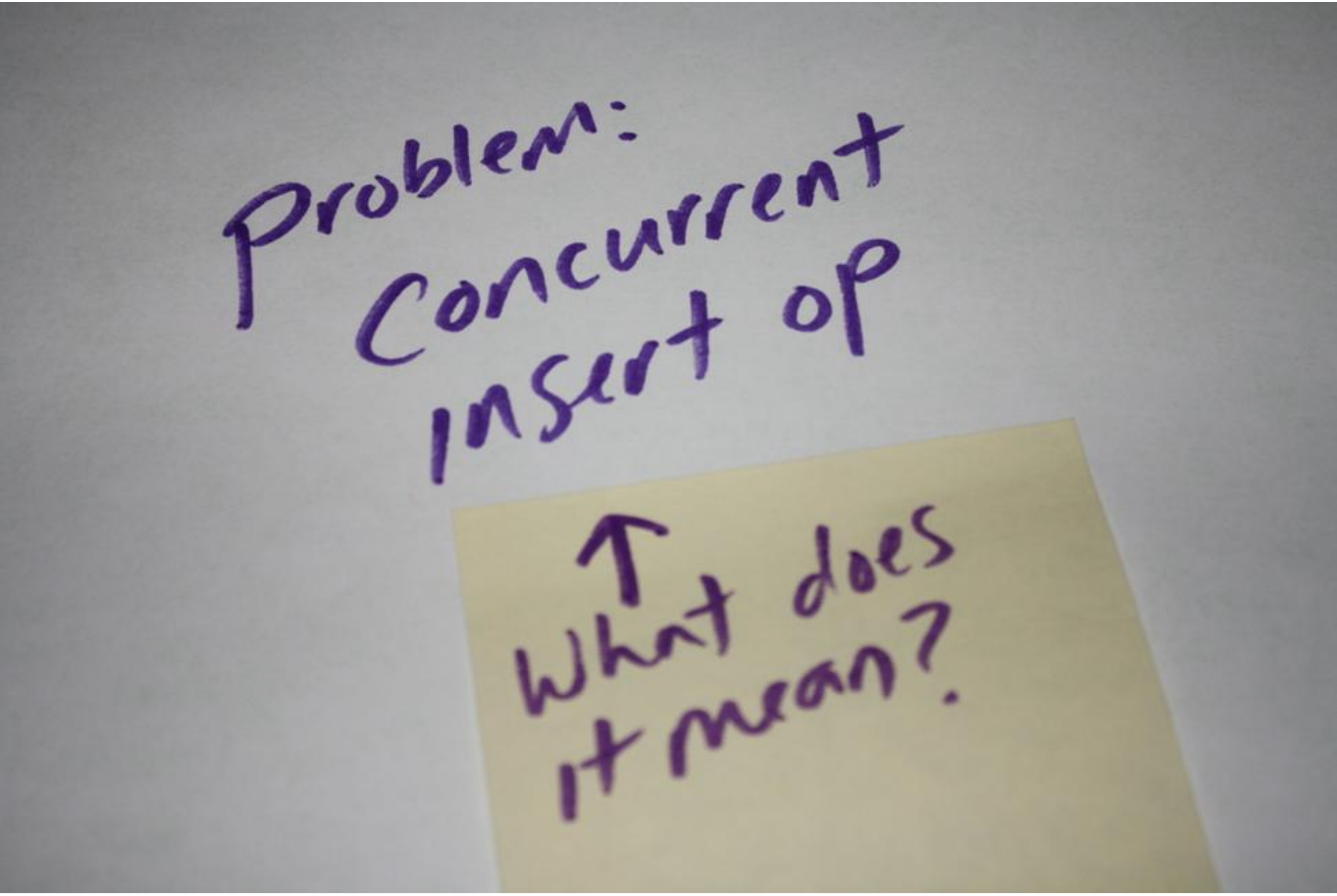
the problem isn't even the same

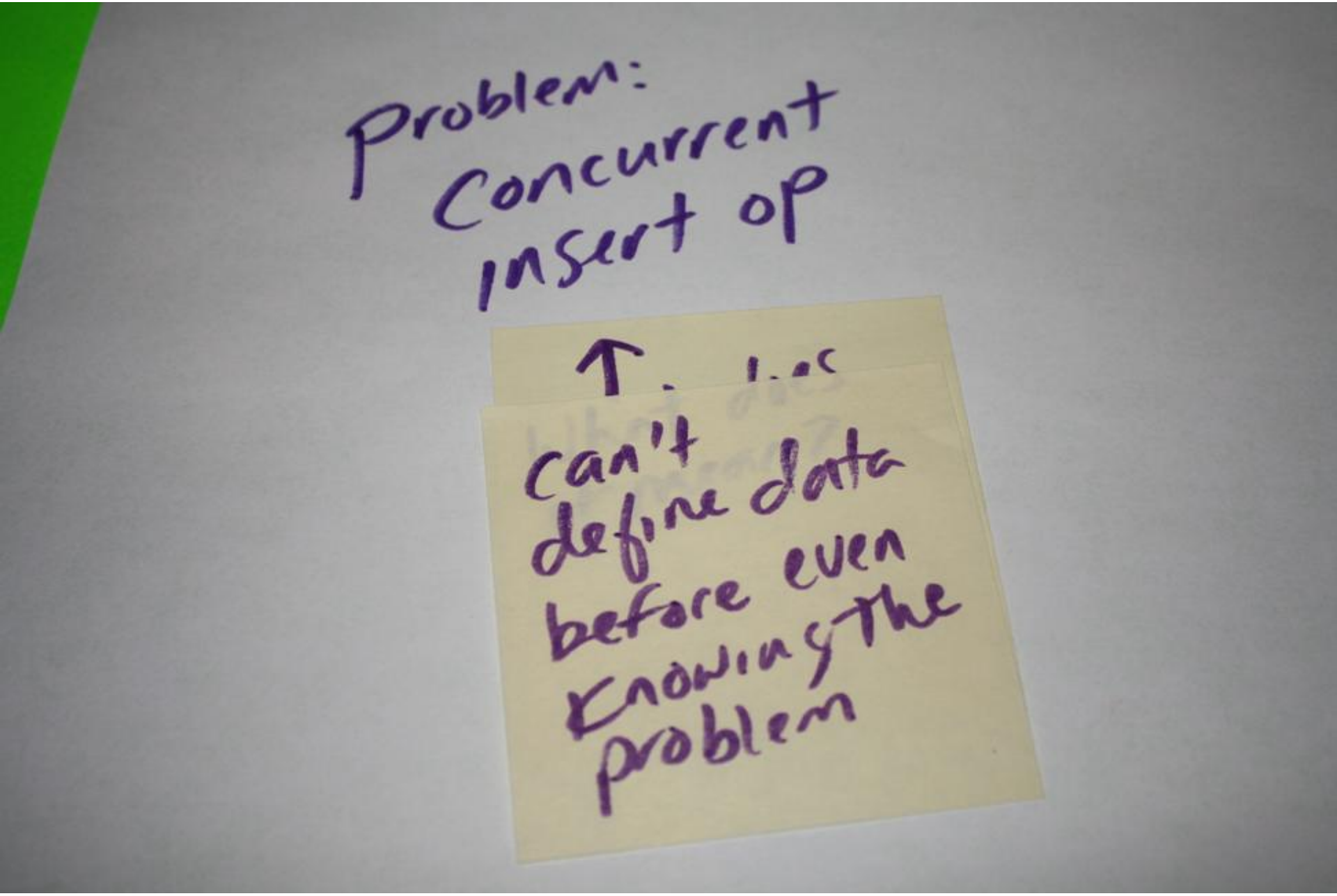


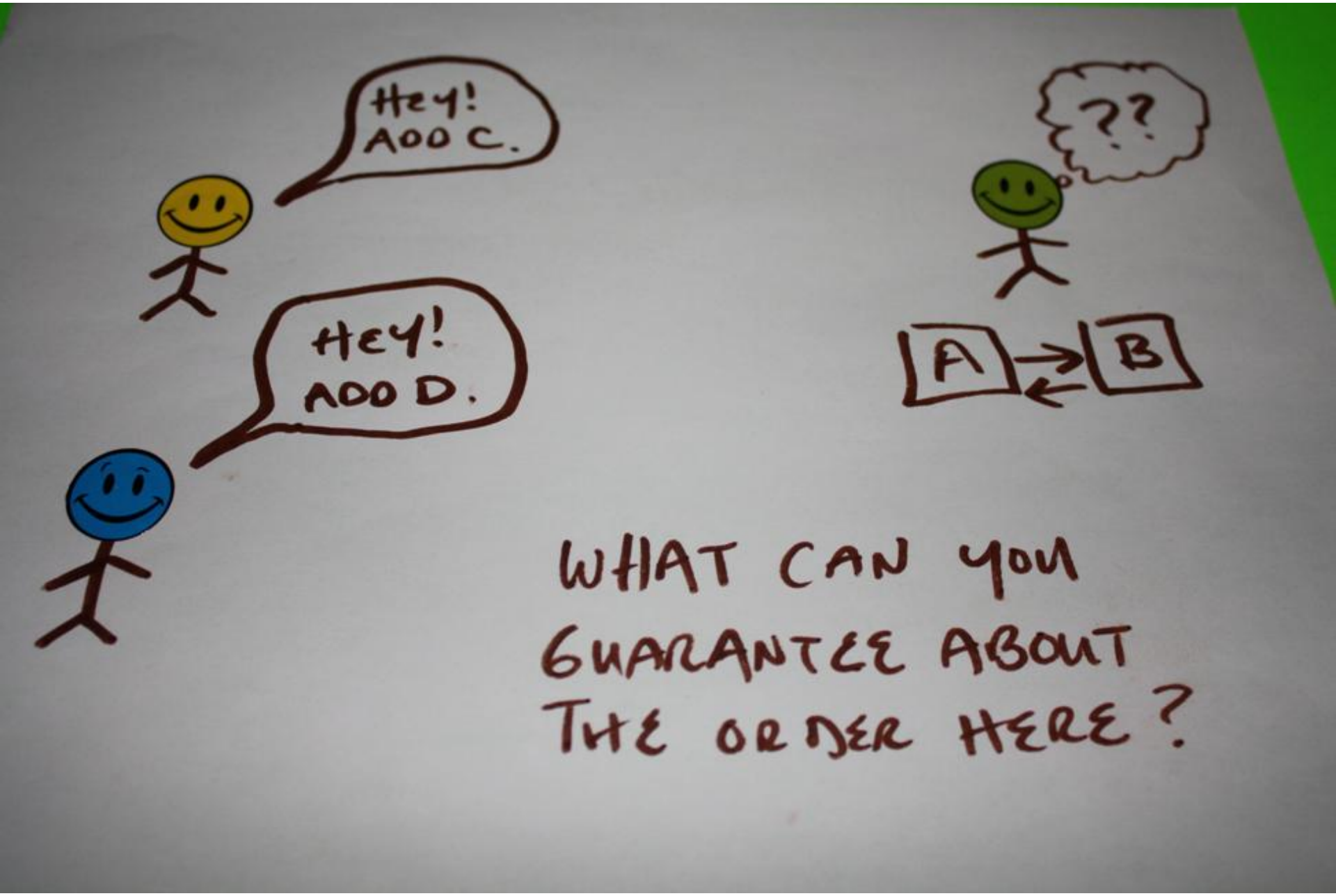
- INSERT (
- INSERT (

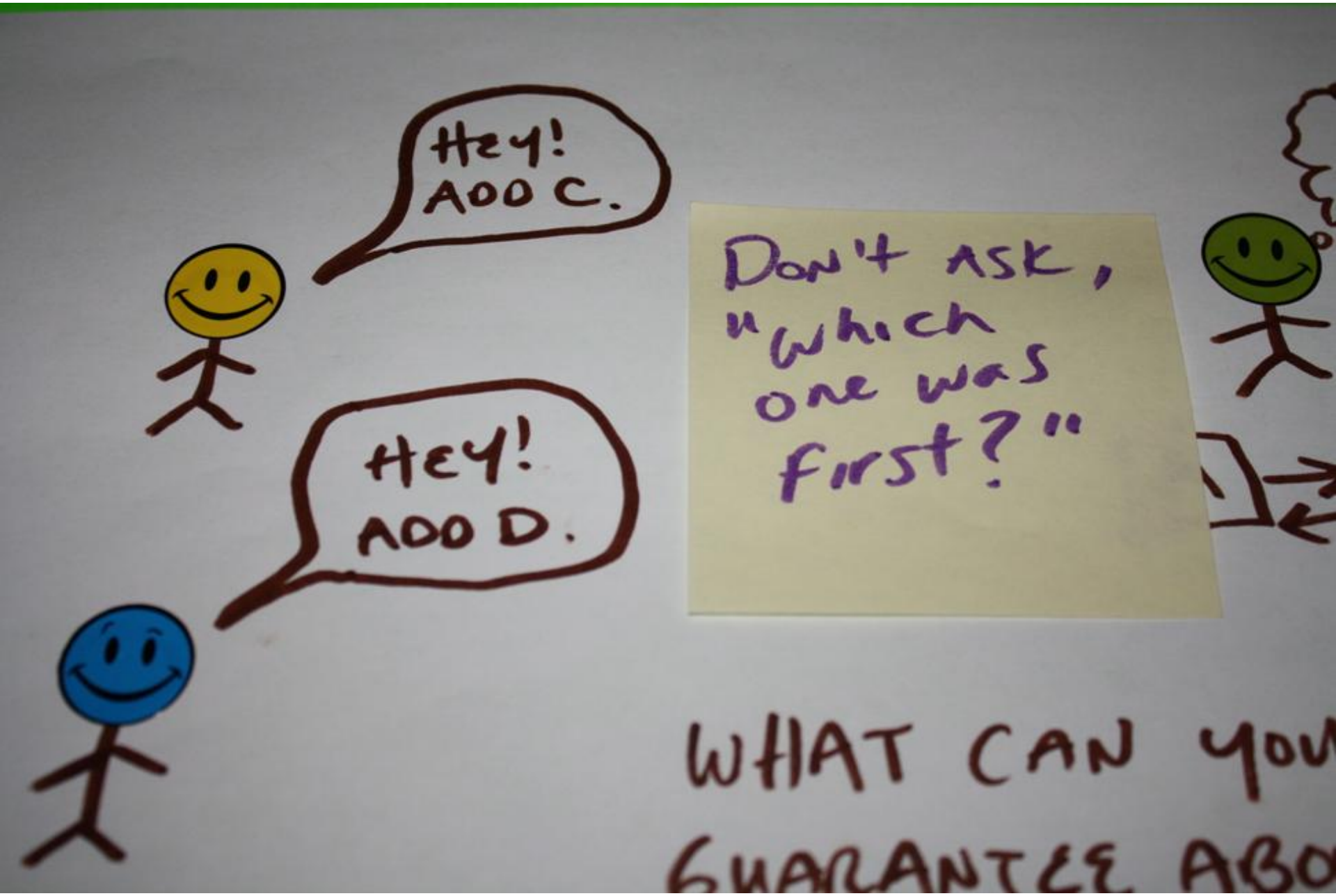

So what part of the Solution would be The Same?
A

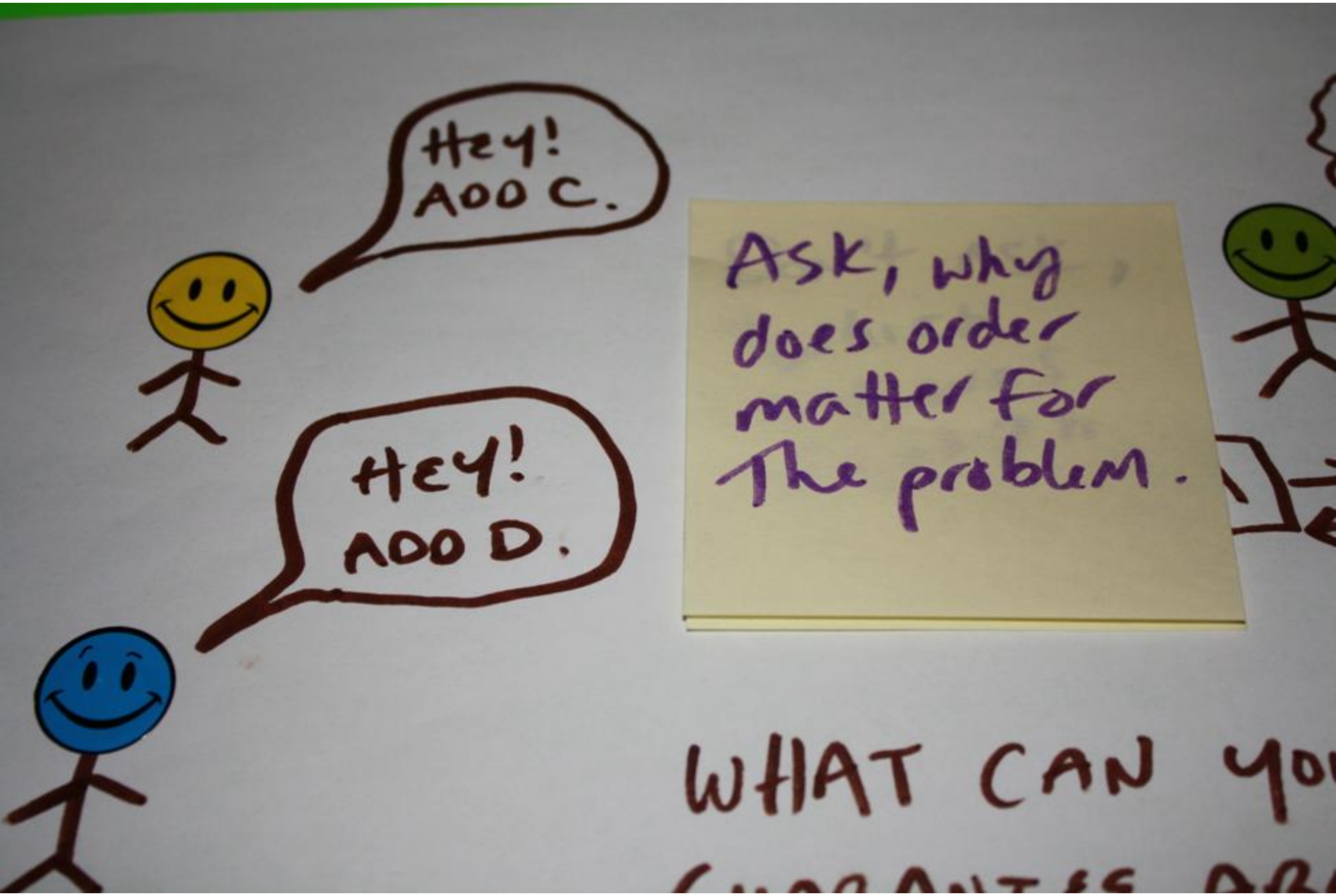

Problem:
Concurrent
insert op


Problem:
Concurrent
insert op
↑
What does
it mean?

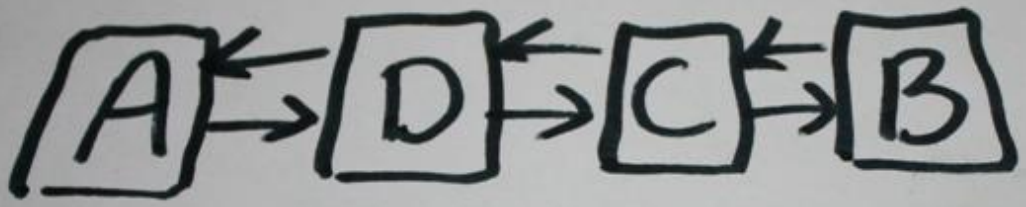

Problem:
Concurrent
insert op
can't
define data
before even
knowing the
problem

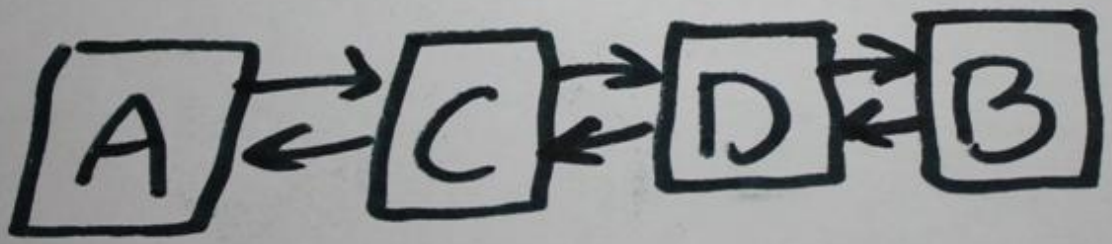

Hey!
ADD C.
Hey!
ADD D.
??
A
B
WHAT CAN YOU
GUARANTEE ABOUT
THE ORDER HERE?


Hey! ADD C.
Hey! ADD D.
Don't ask, "which one was first?"
WHAT CAN YOU
GUARANTEE ABO

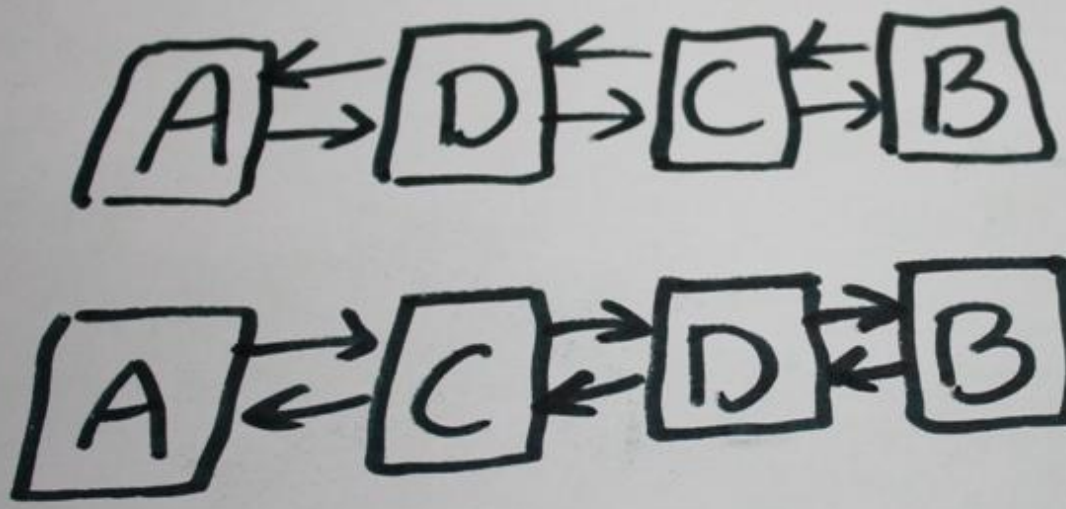

Hey!
ADD C.
Hey!
ADD D.
Ask, why does order matter for The problem.
WHAT CAN YOU

WHICH ORDER IS "CORRECT" ?



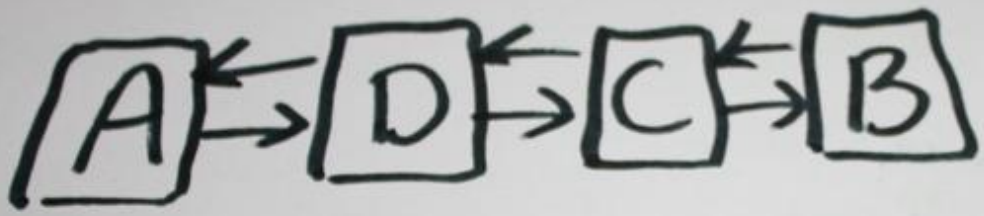


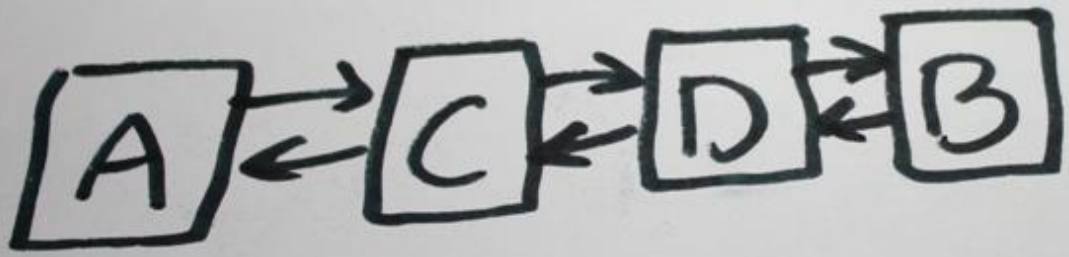
WHICH ORDER IS "CORRECT"?



BOTH!

WHICH ORDER IS "CORRECT"?

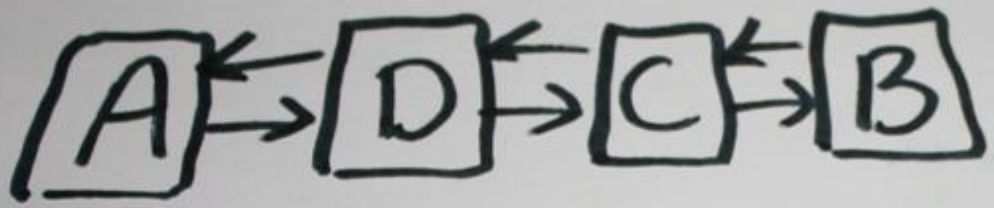




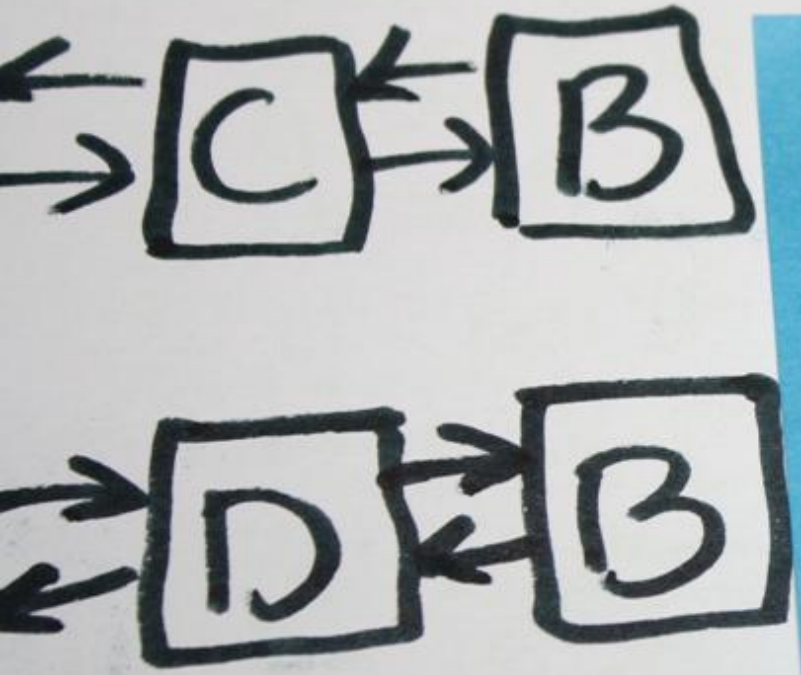
BOTH!

NEITHER!

WHICH ORDER IS "CORRECT"?



DEPENDS ON CONTEXT.

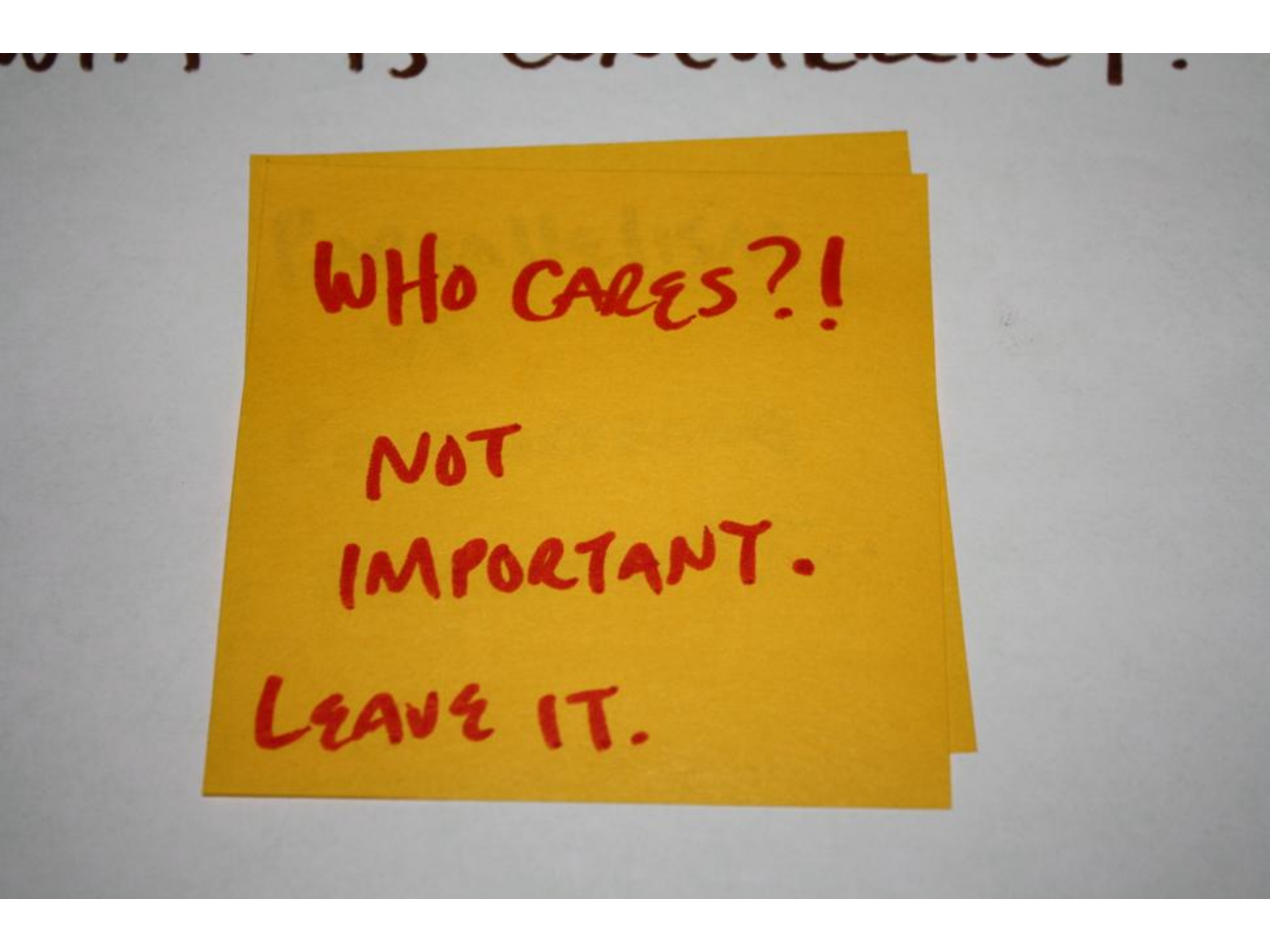


"CONTEXT" INCLUDES EXPLICIT ORDERING RULES.

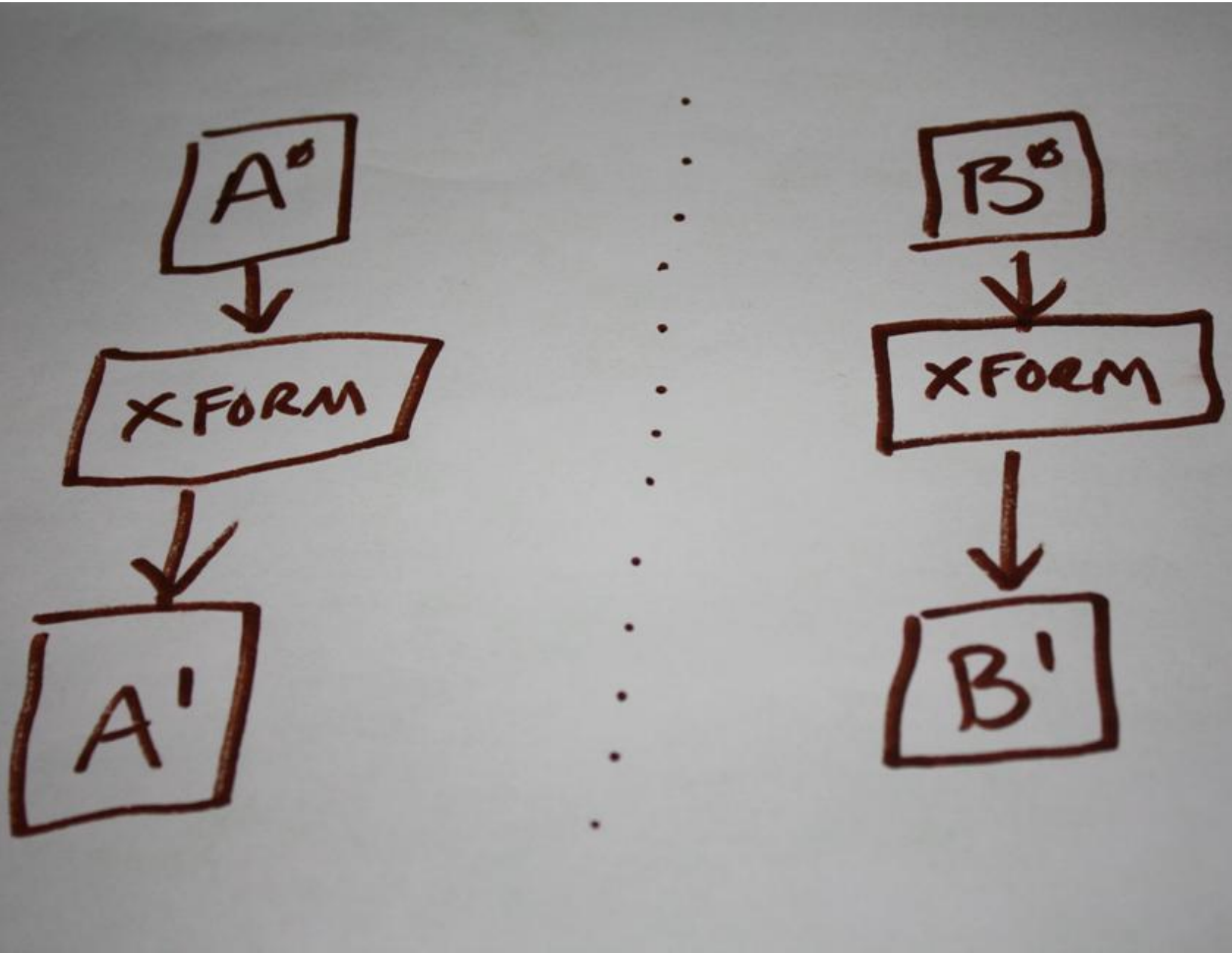
BUT FIRST...

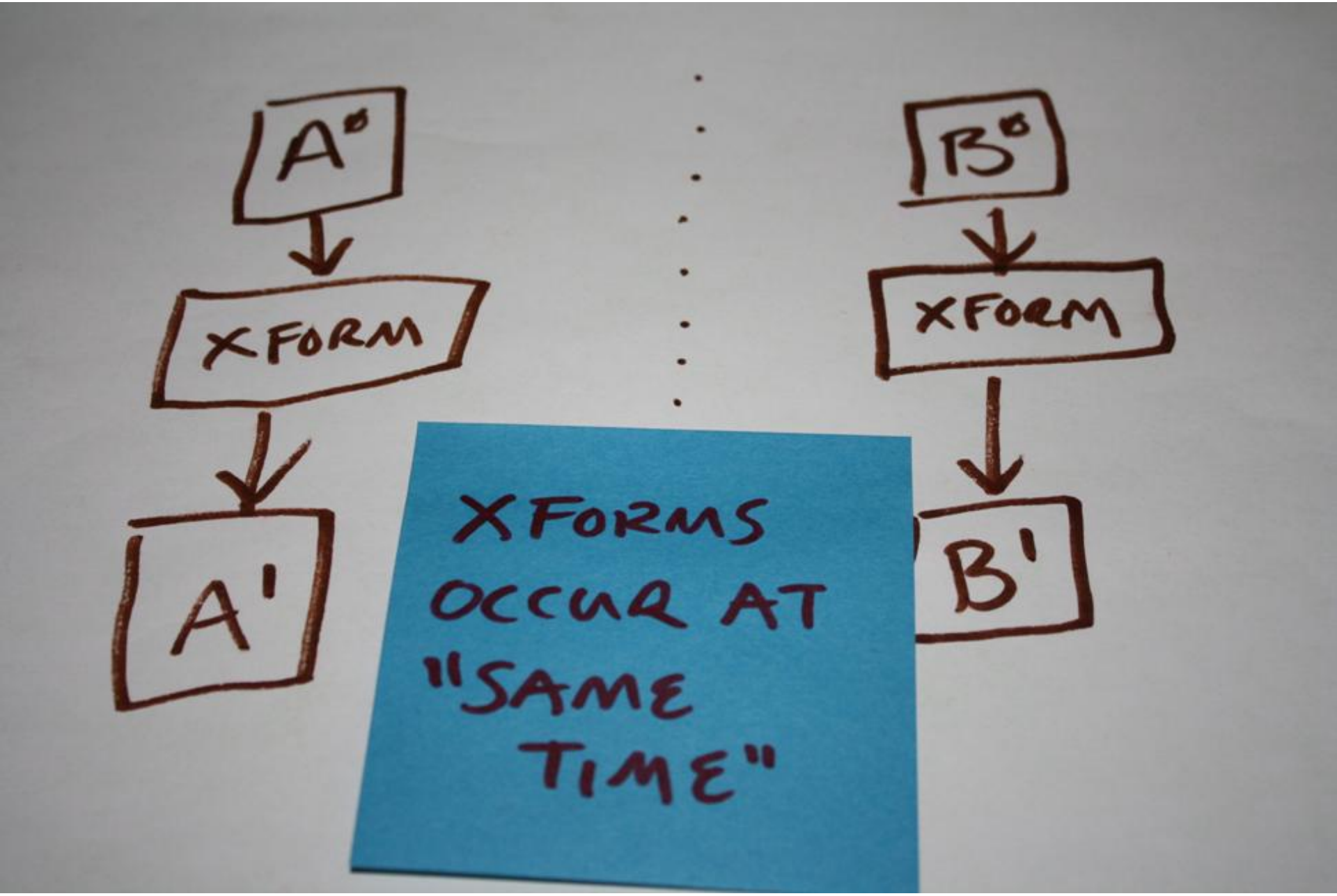
WHAT IS CONCURRENCY?

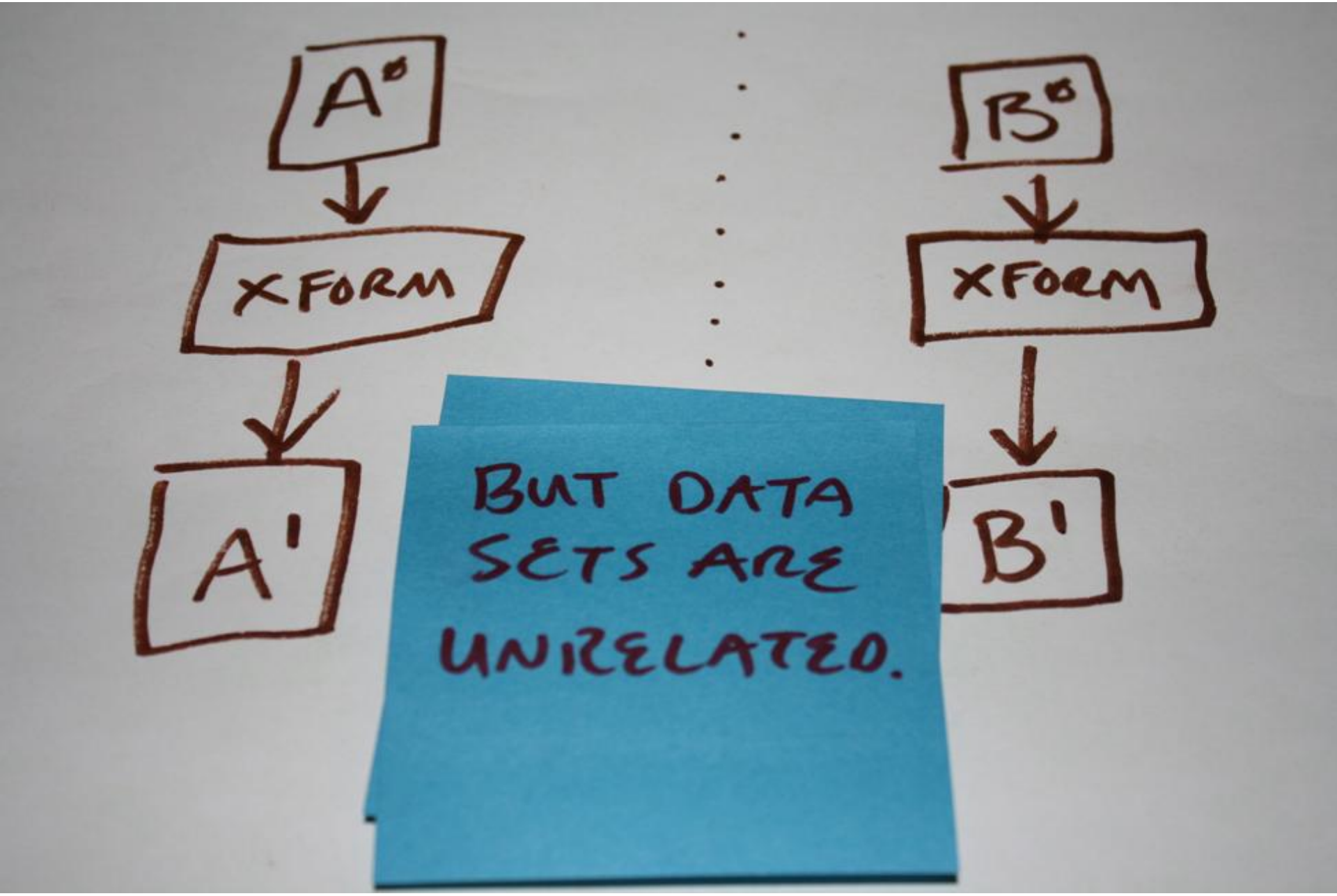

WHAT IS CONCURRENCY?
PARALLELISM
VS.
CONCURRENCY
ARGUMENTS...

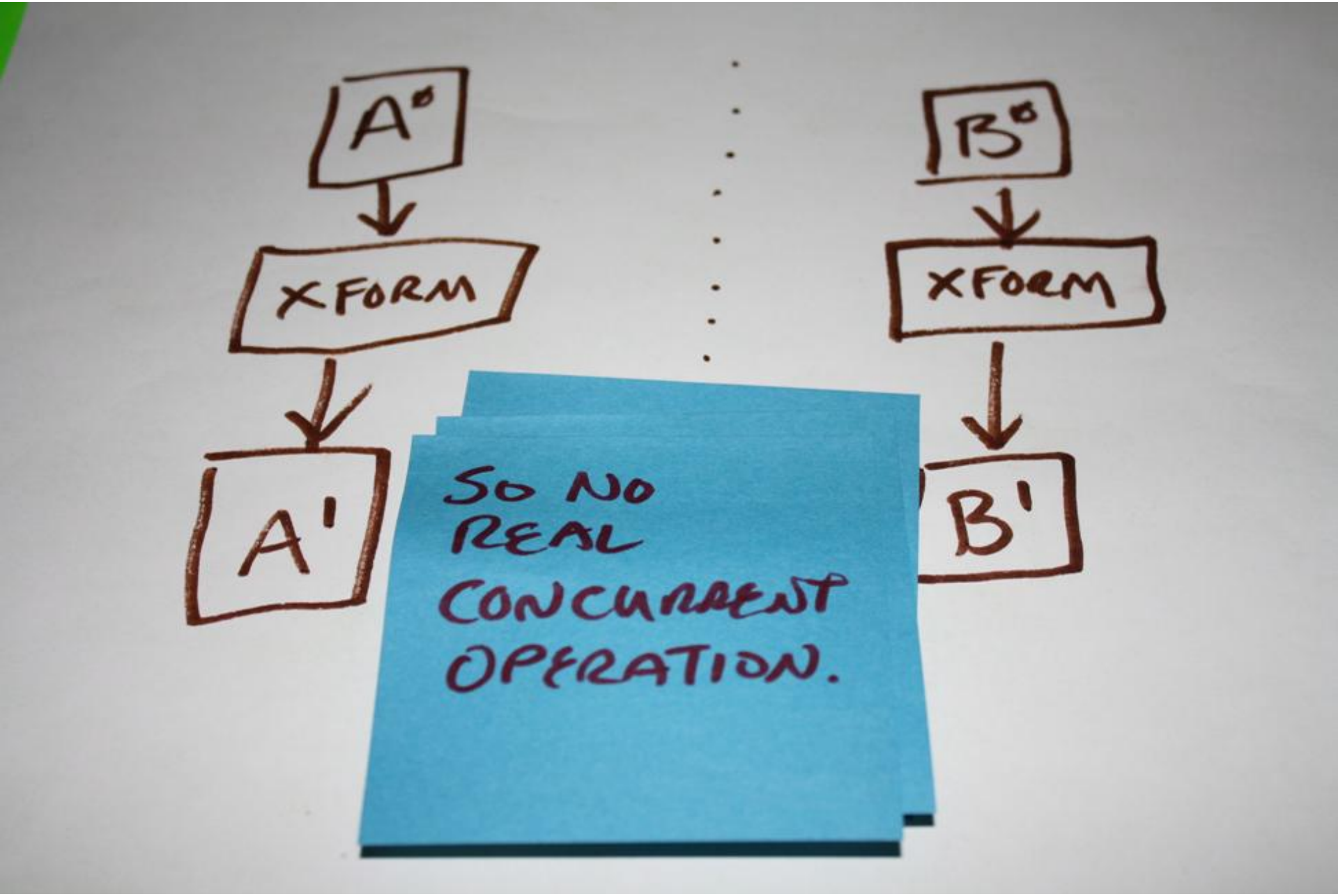

WHO CARES?!
NOT
IMPORTANT.
LEAVE IT.

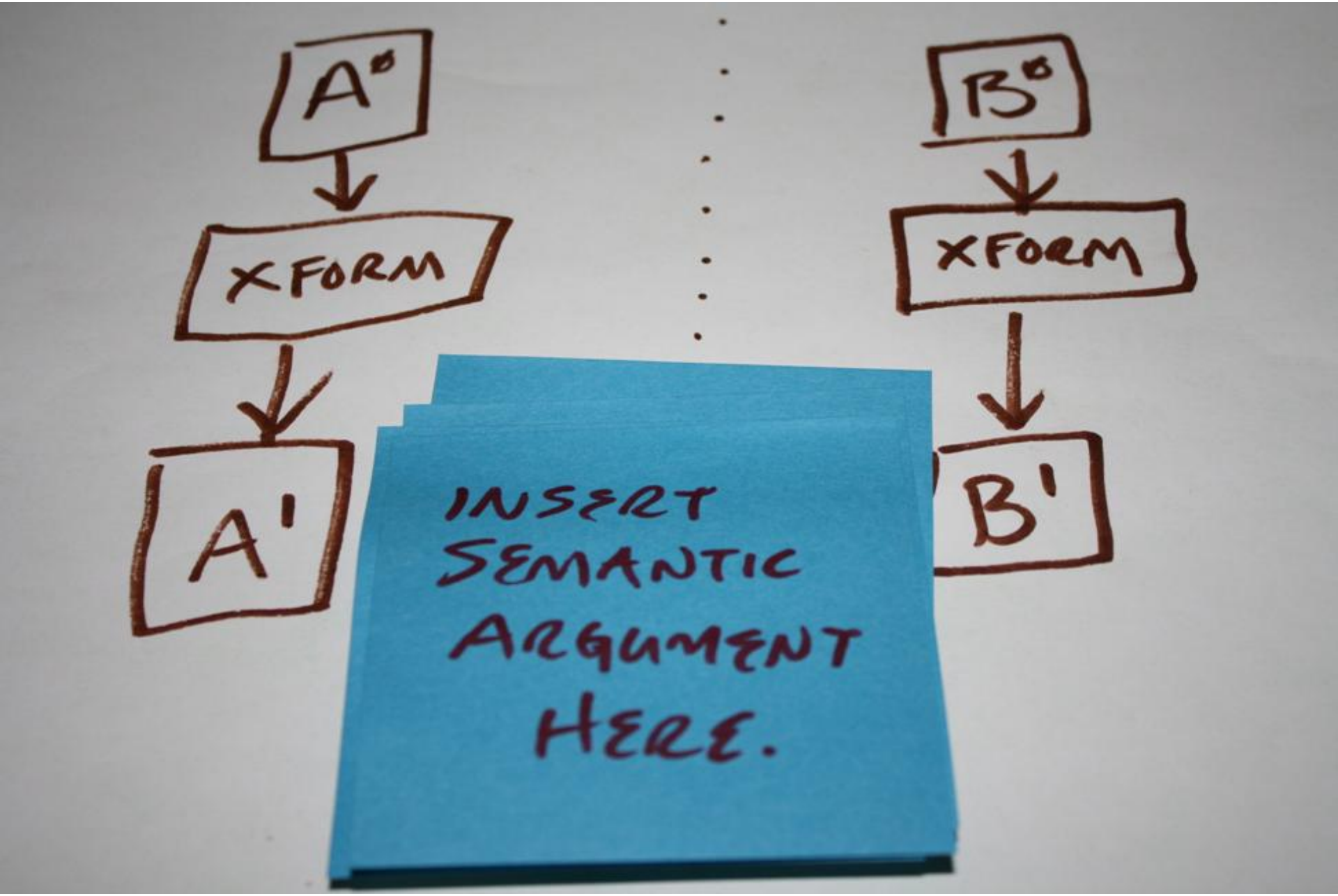
CONCURRENCY IS TRANSFORMATION OF SHARED DATA SET.

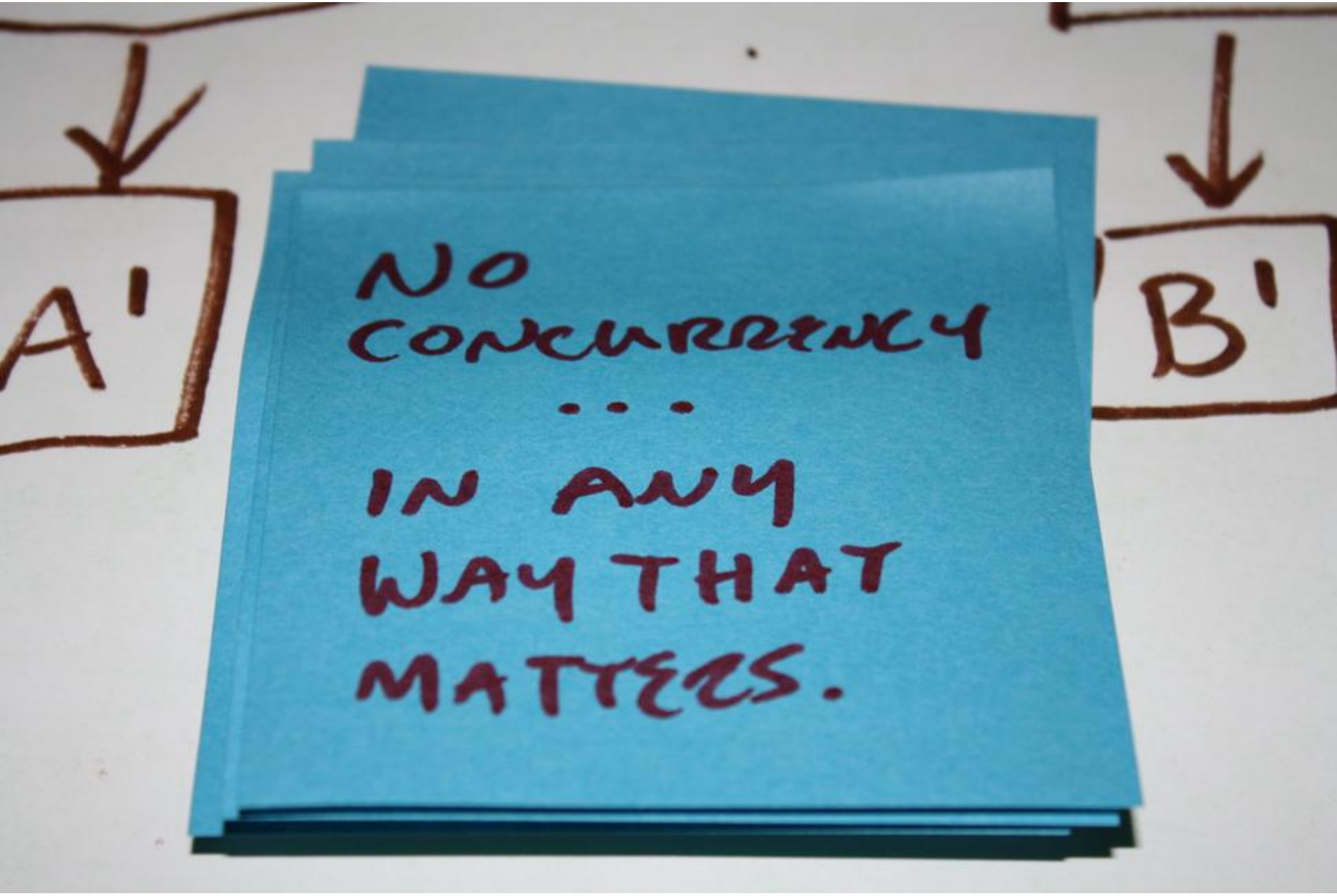

A°
XFORM
A'
B°
XFORM
B'

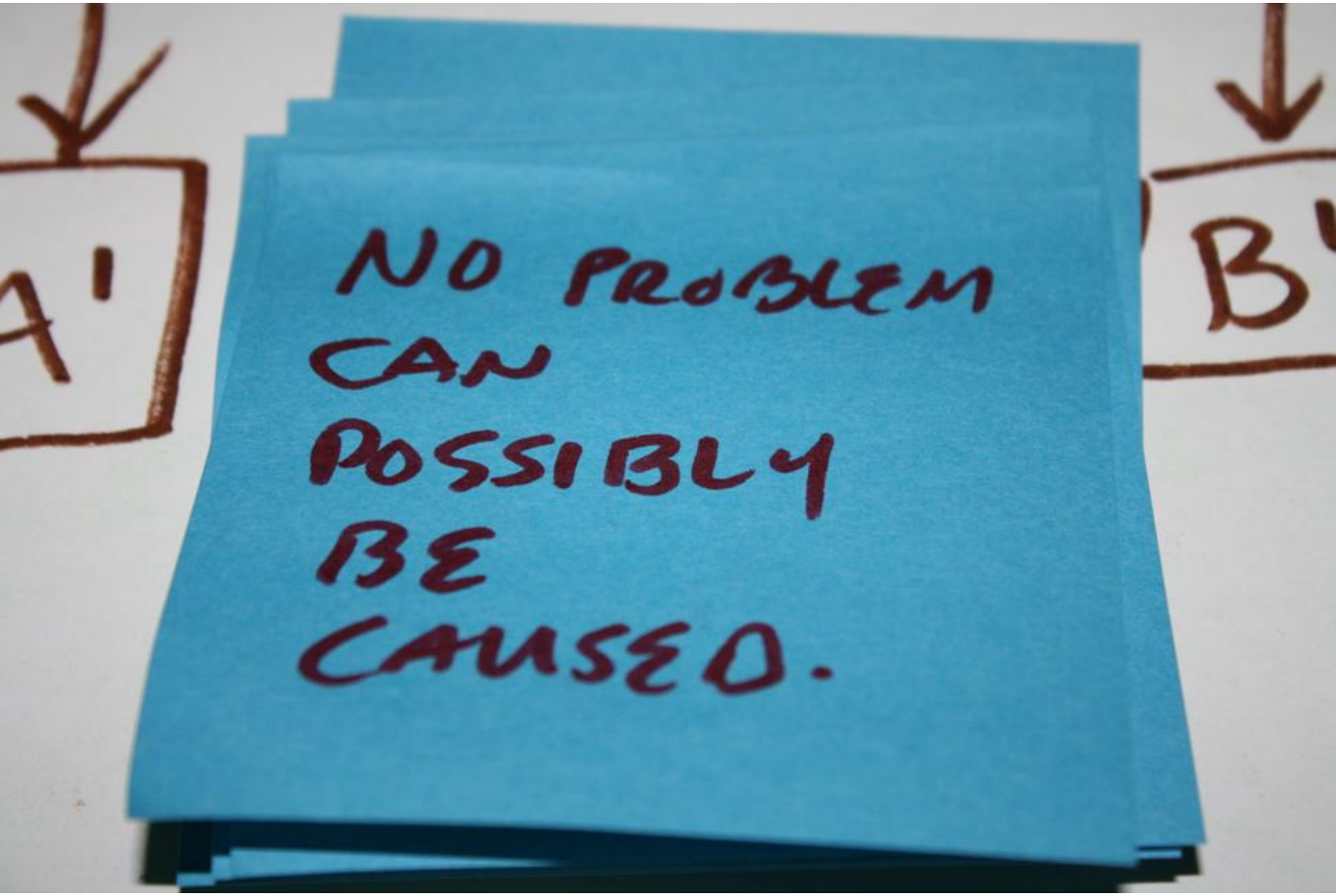

A°
XFORM
A'
B°
XFORM
B'
XFORMS
OCCUR AT
"SAME
TIME"

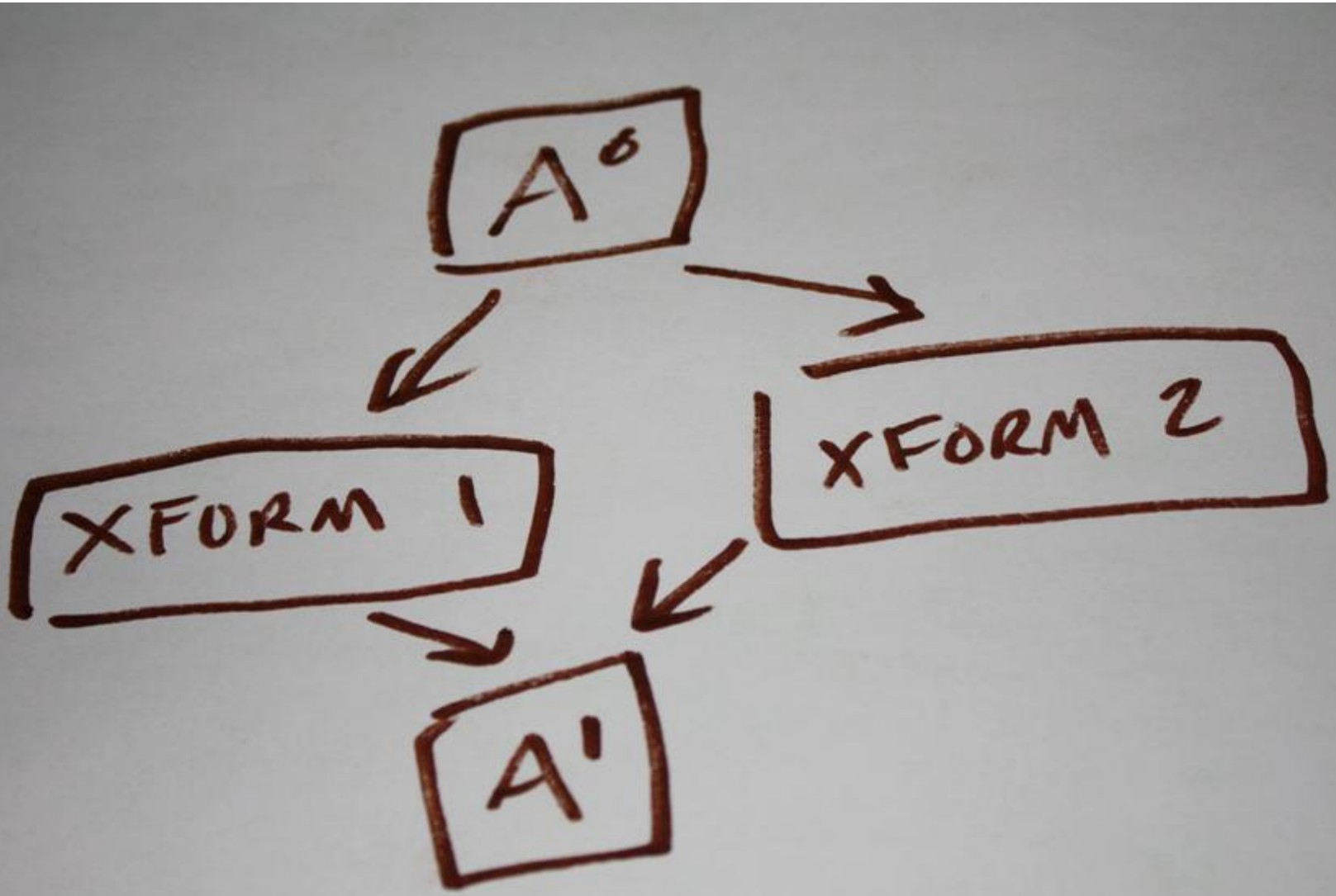

A°
XFORM
A'
B°
XFORM
B'
BUT DATA SETS ARE UNRELATED.


A°
XFORM
A'
B°
XFORM
B'
SO NO
REAL
CONCURRENT
OPERATION.

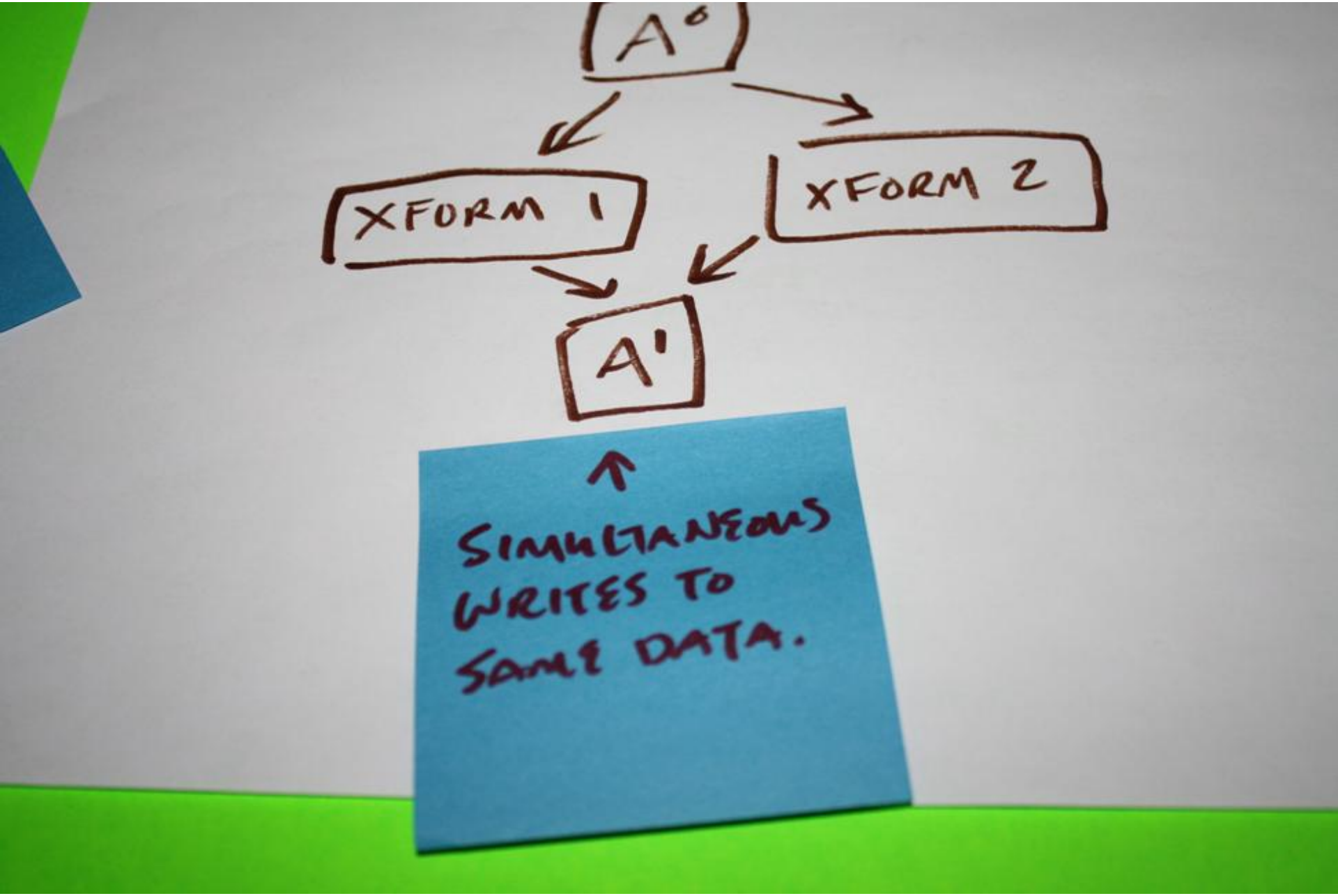

A°
XFORM
A'
B°
XFORM
B'
INSERT
SEMANTIC
ARGUMENT
HERE.

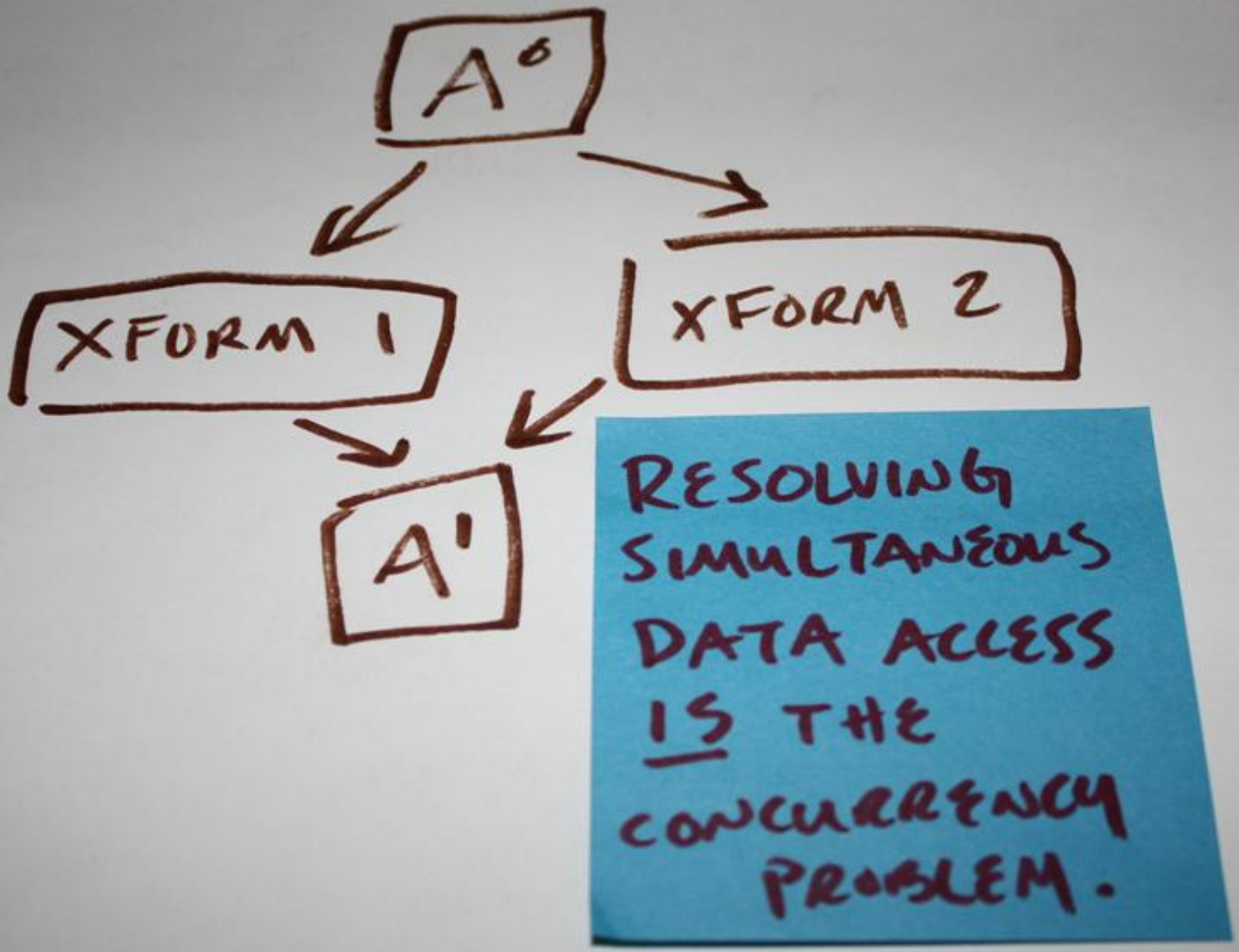

A'
B'
NO CONCURRENCY ...
IN ANY WAY THAT MATTERS.

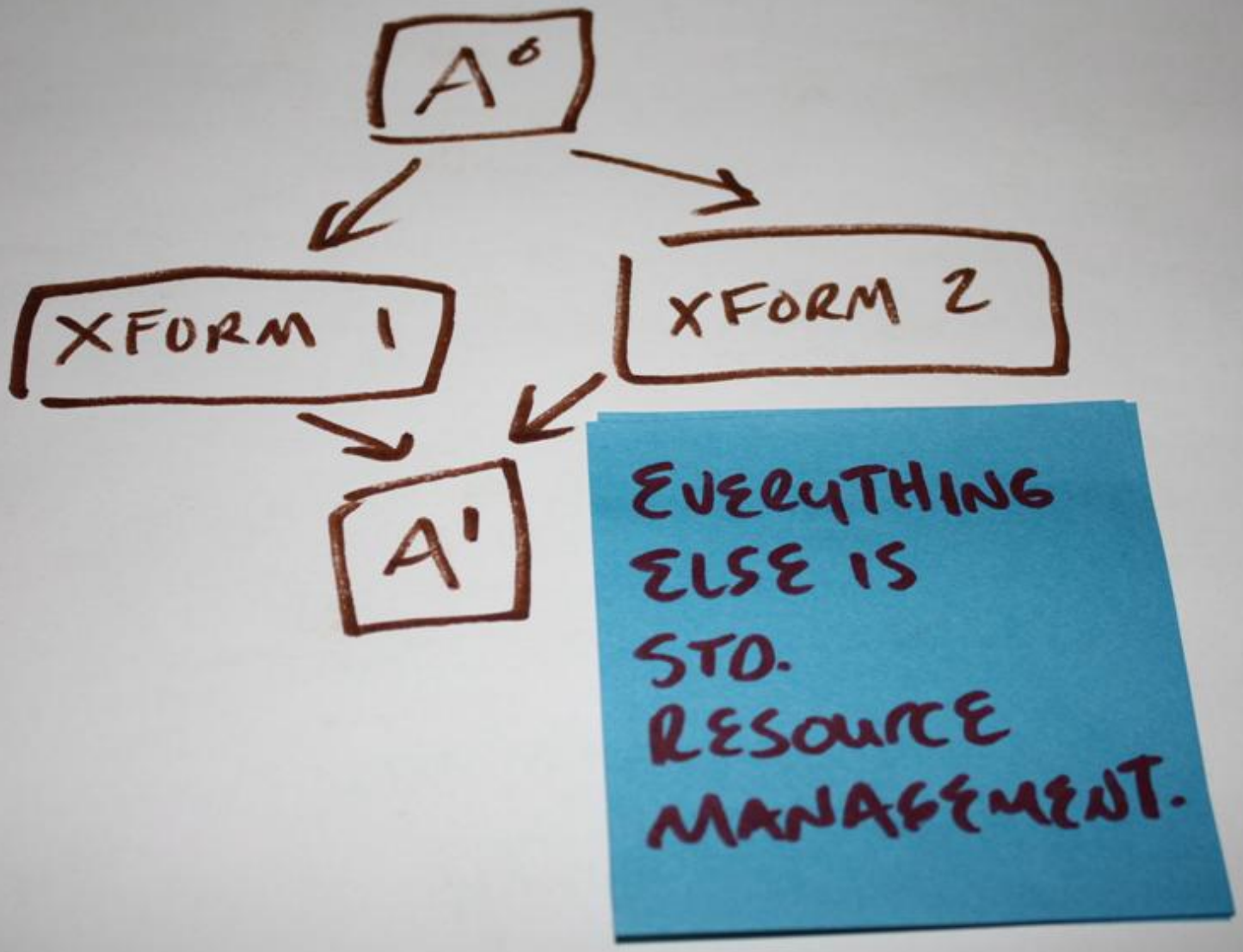

A'
B'
NO PROBLEM
CAN
POSSIBLY
BE
CAUSED.

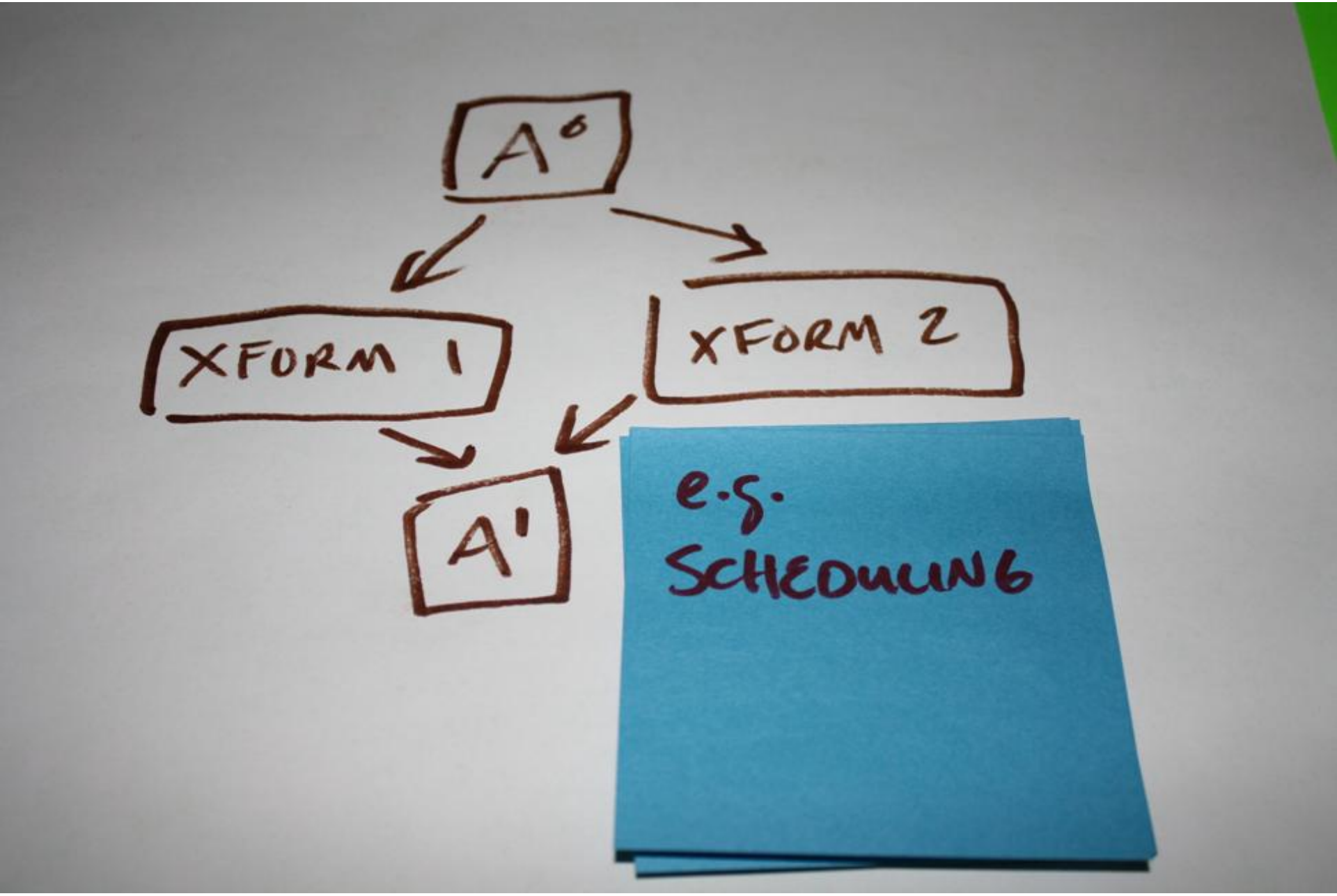

A⁰
XFORM 1
XFORM 2
A'


A°
XFOR
ORM 2
SIMULTANEOUS
READS FROM
SAME DATA


A°
XFORM 1
XFORM 2
A'
SIMULTANEOUS
WRITES TO
SAME DATA.


A°
XFORM 1
XFORM 2
A'
RESOLVING SIMULTANEOUS DATA ACCESS IS THE CONCURRENCY PROBLEM.


A⁰
XFORM 1
XFORM 2
A'
EVERYTHING
ELSE IS
STD.
RESOURCE
MANAGEMENT.


A°
XFORM 1
XFORM 2
A'
e.g.
SCHEDULING

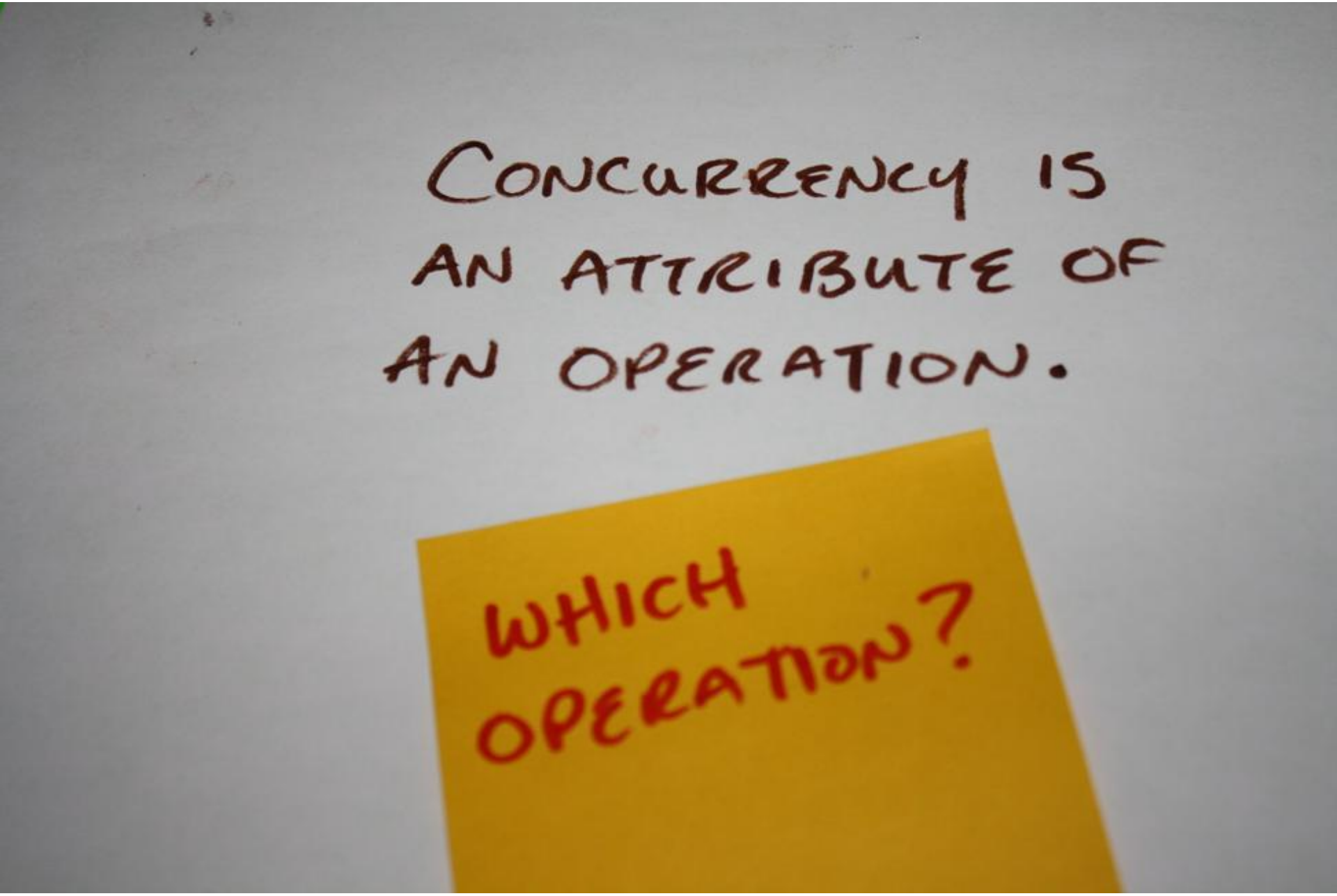
FIRST - VARIOUS PARALLELISM

- INSTRUCTION LEVEL
- MULTI - THREADING
- MULTI - CORE, SHARED MEM
- MULT - CORE, INDEPENDENT MEM
- MULT - MACHINES

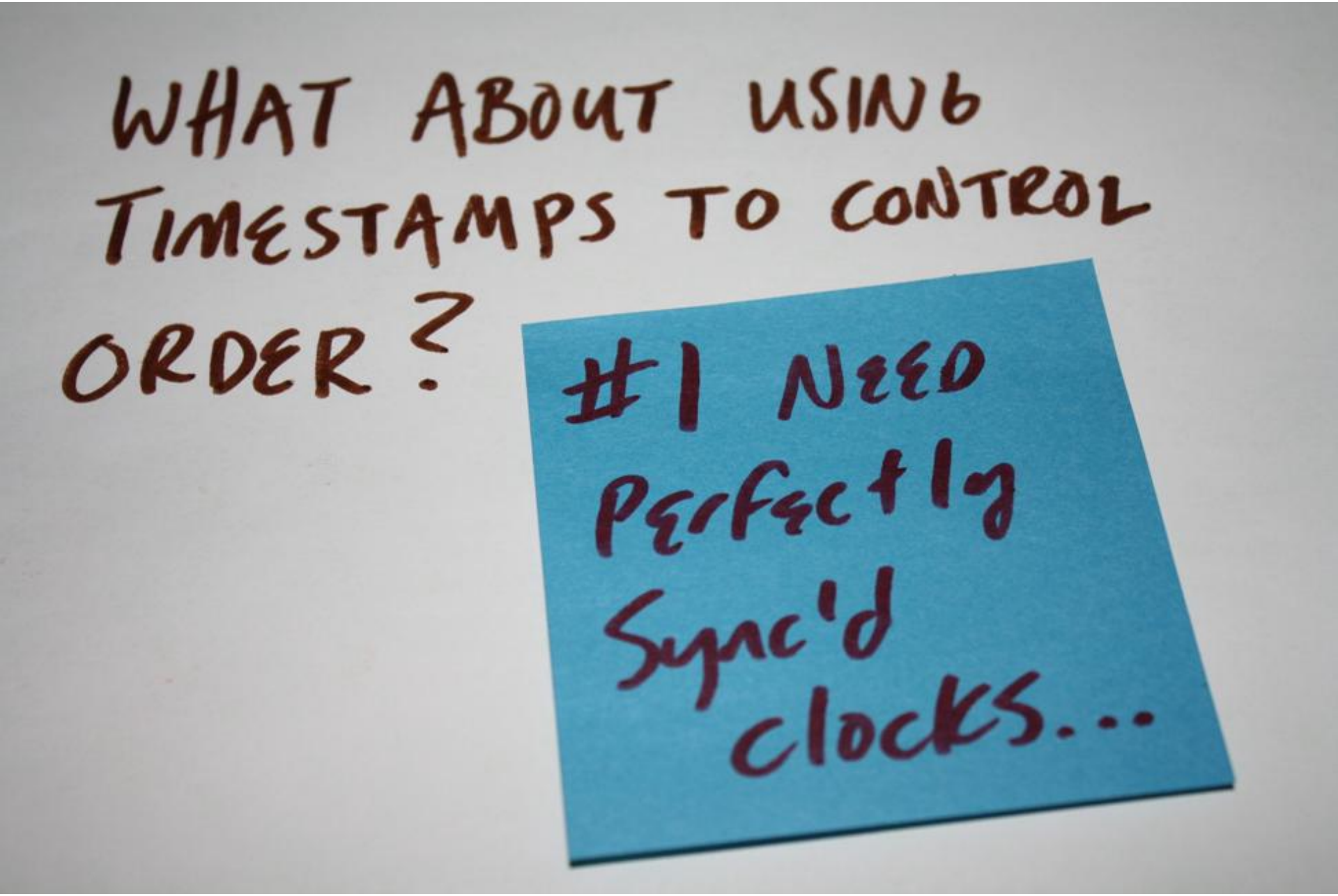
WHAT DO ALL OF THESE HAVE IN COMMON?

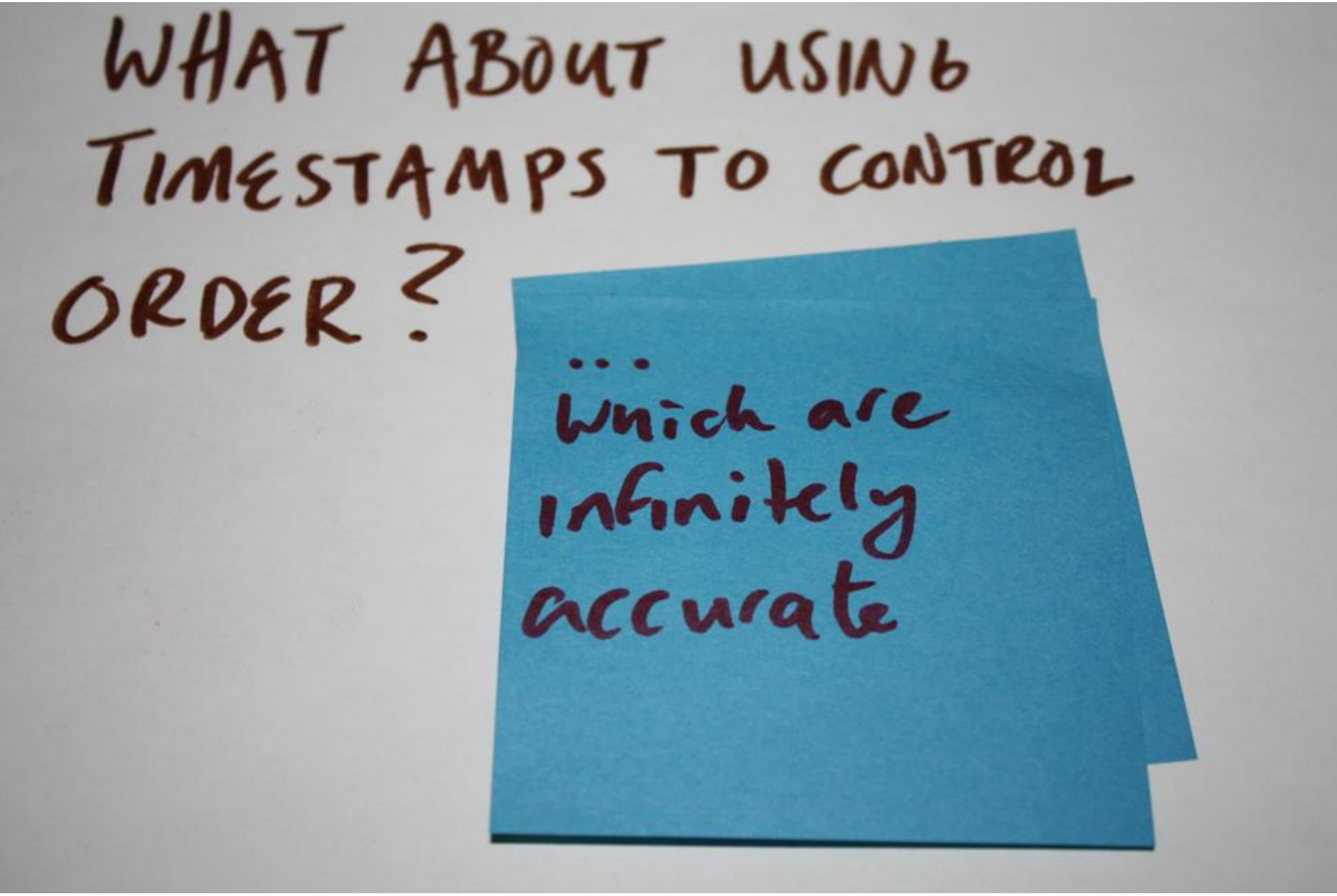
SIMULTANEOUS
XFORM OF
SAME DATA FILE

CONCURRENCY IS AN ATTRIBUTE OF AN OPERATION.

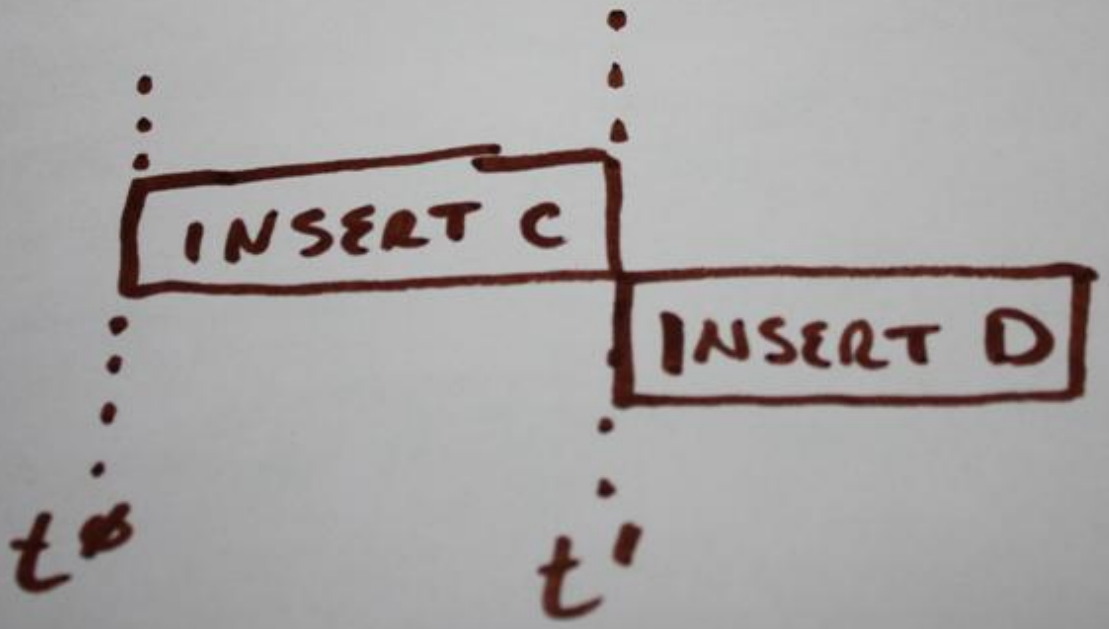

CONCURRENCY IS
AN ATTRIBUTE OF
AN OPERATION.
WHICH
OPERATION?

WHAT ABOUT USING TIMESTAMPS TO CONTROL ORDER?

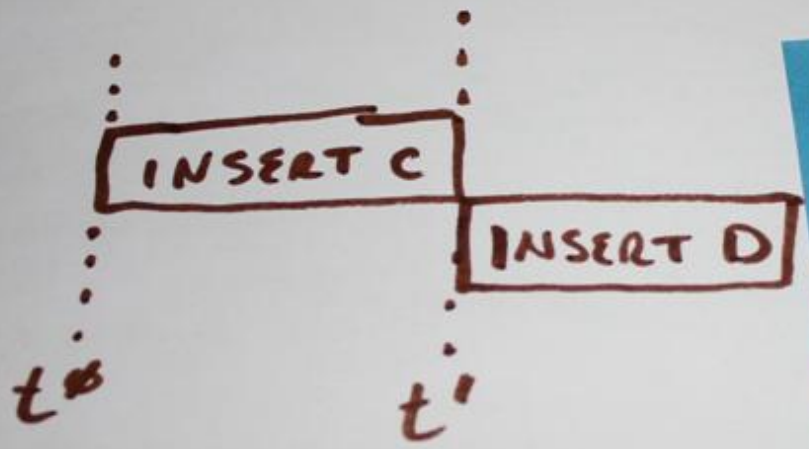

WHAT ABOUT USING TIMESTAMPS TO CONTROL ORDER?
#1 NEED
Perfectly
Sync'd
clocks...


WHAT ABOUT USING TIMESTAMPS TO CONTROL ORDER?
...
which are infinitely accurate

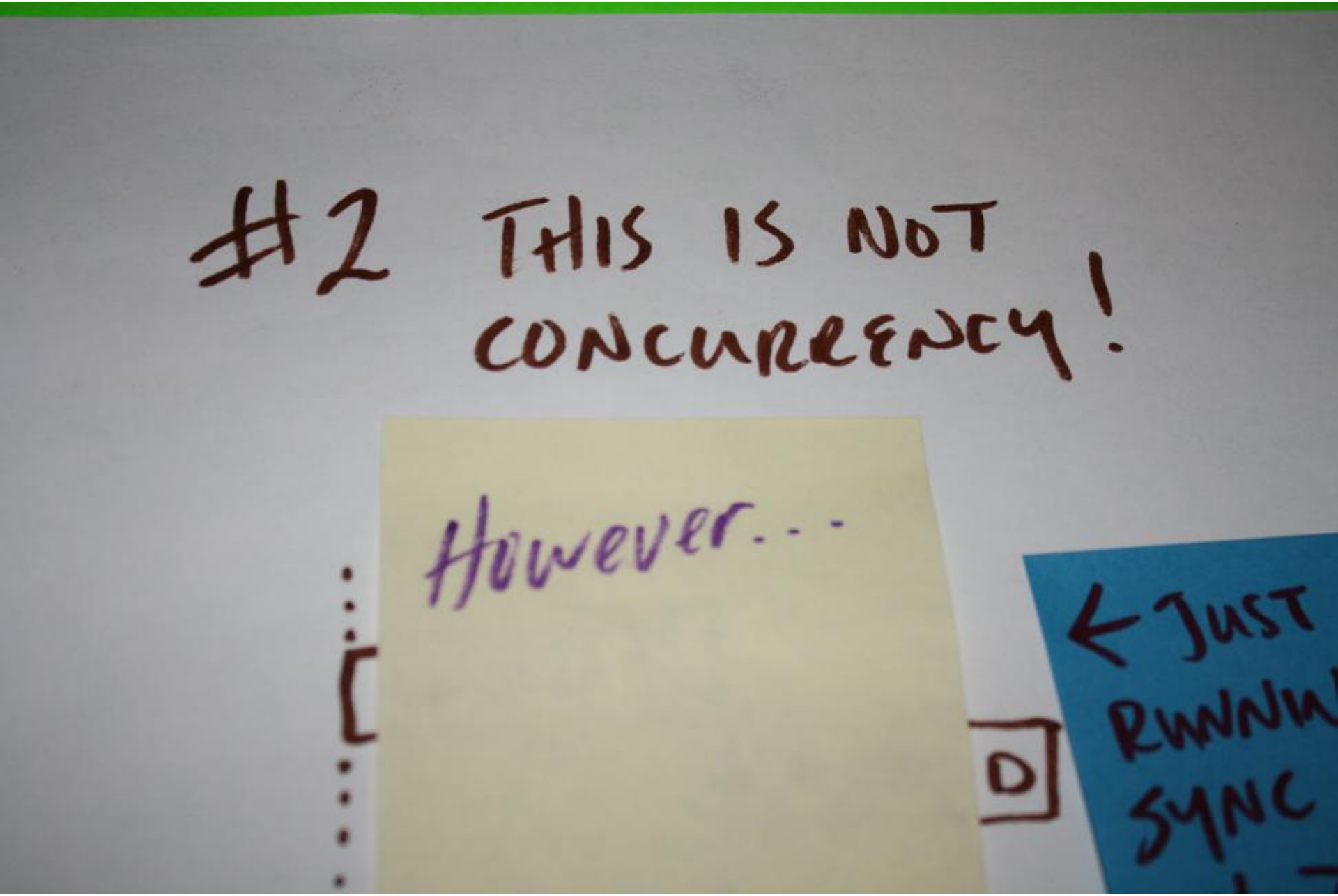
#2 THIS IS NOT CONCURRENCY!

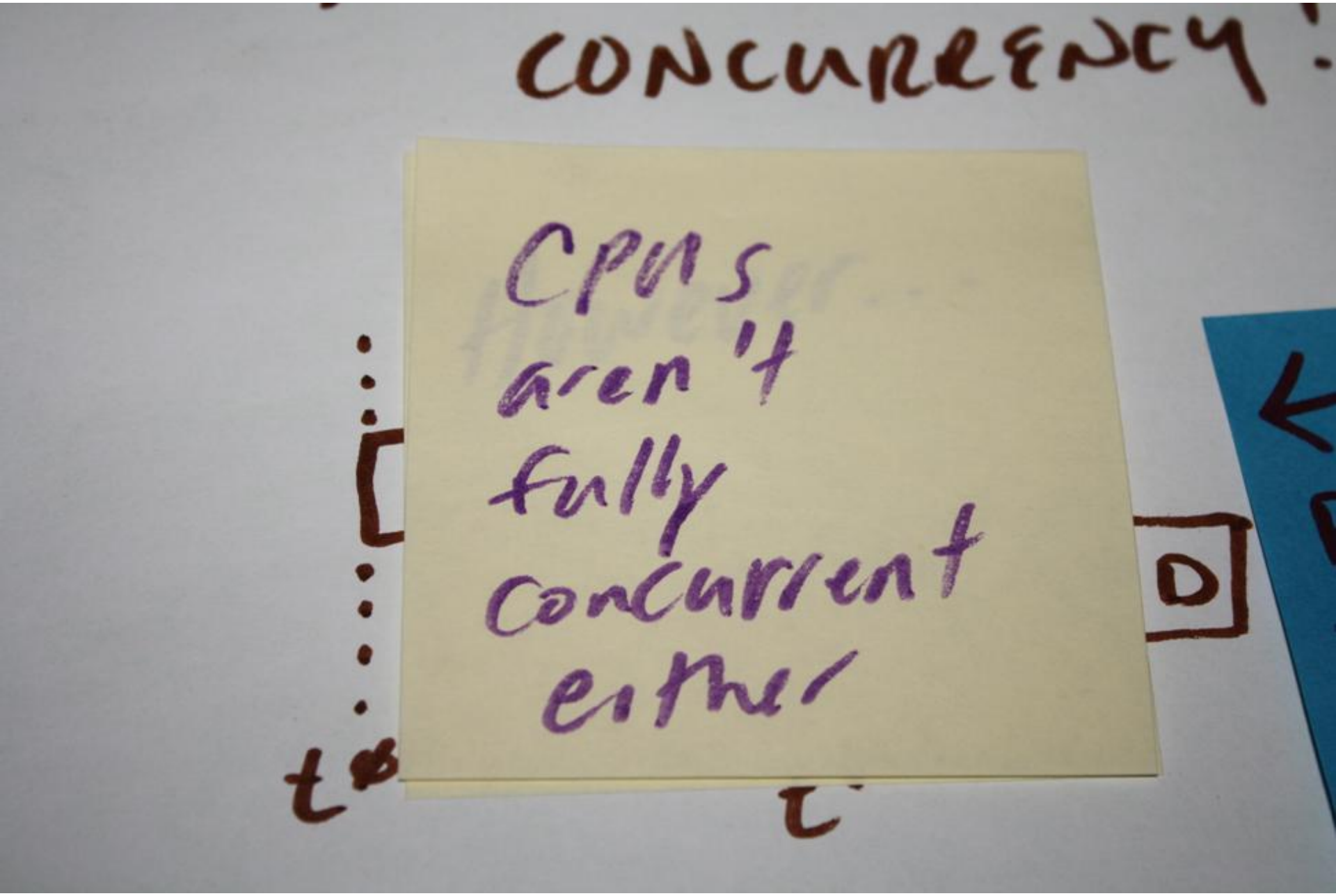


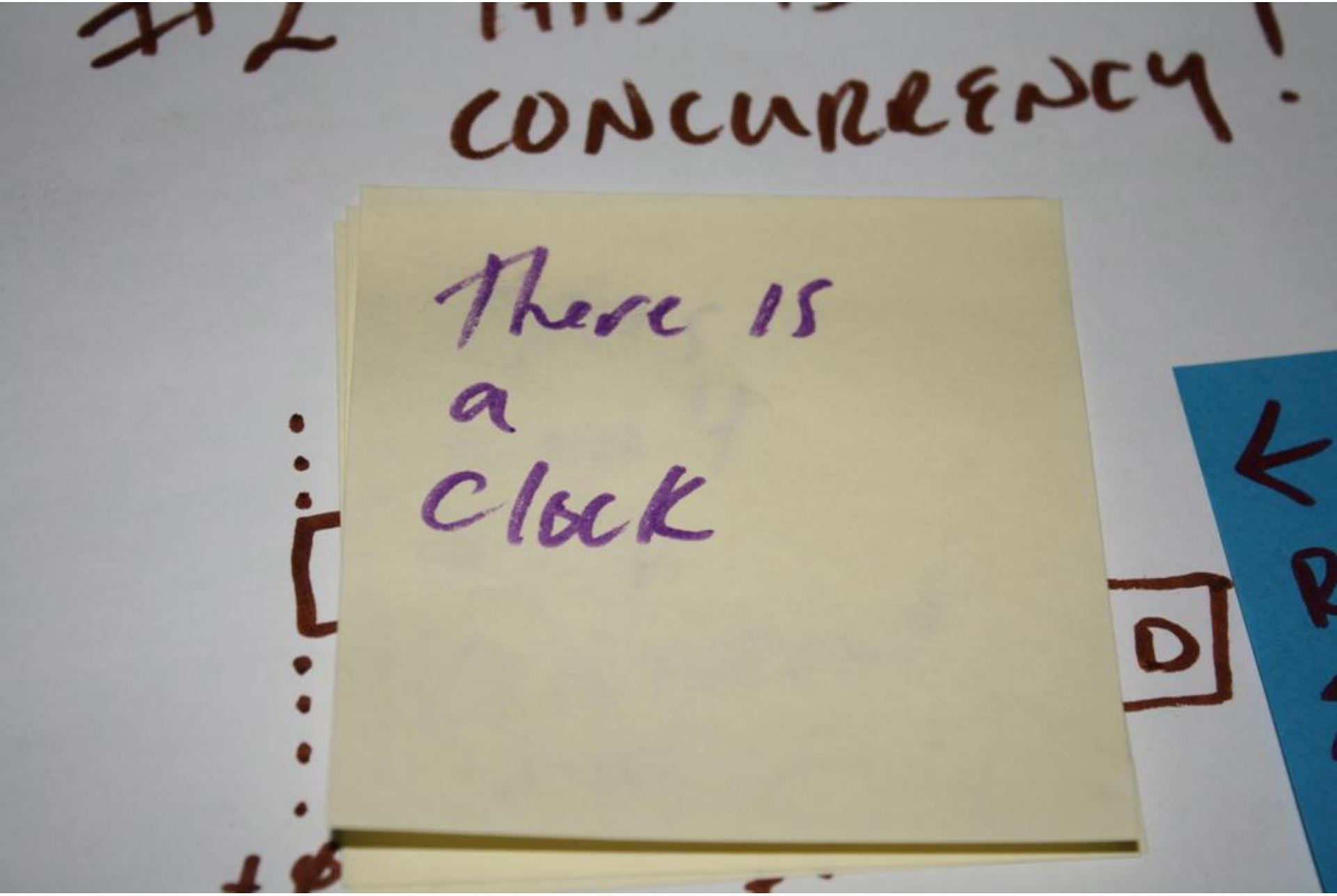
#2 THIS IS NOT CONCURRENCY!

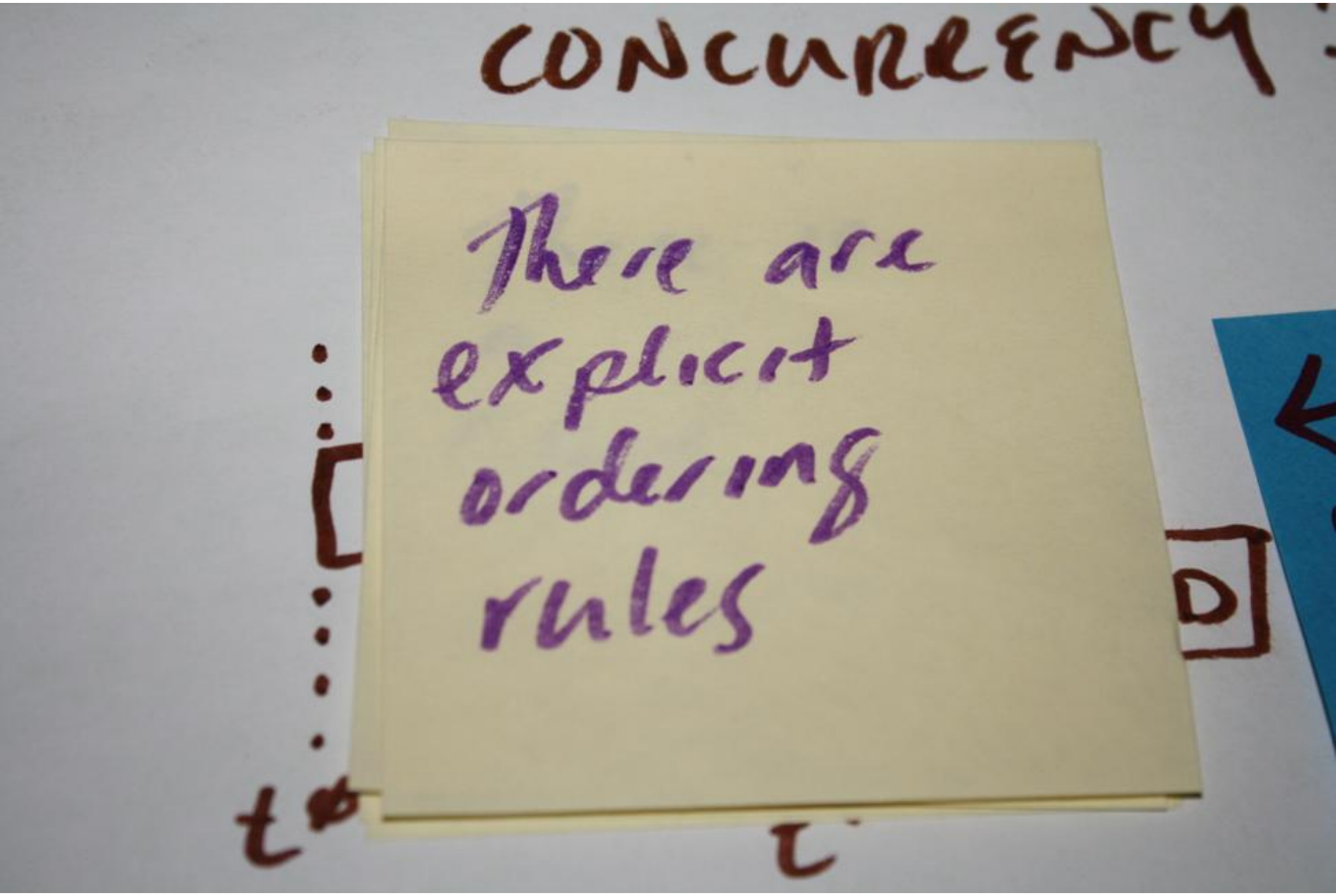


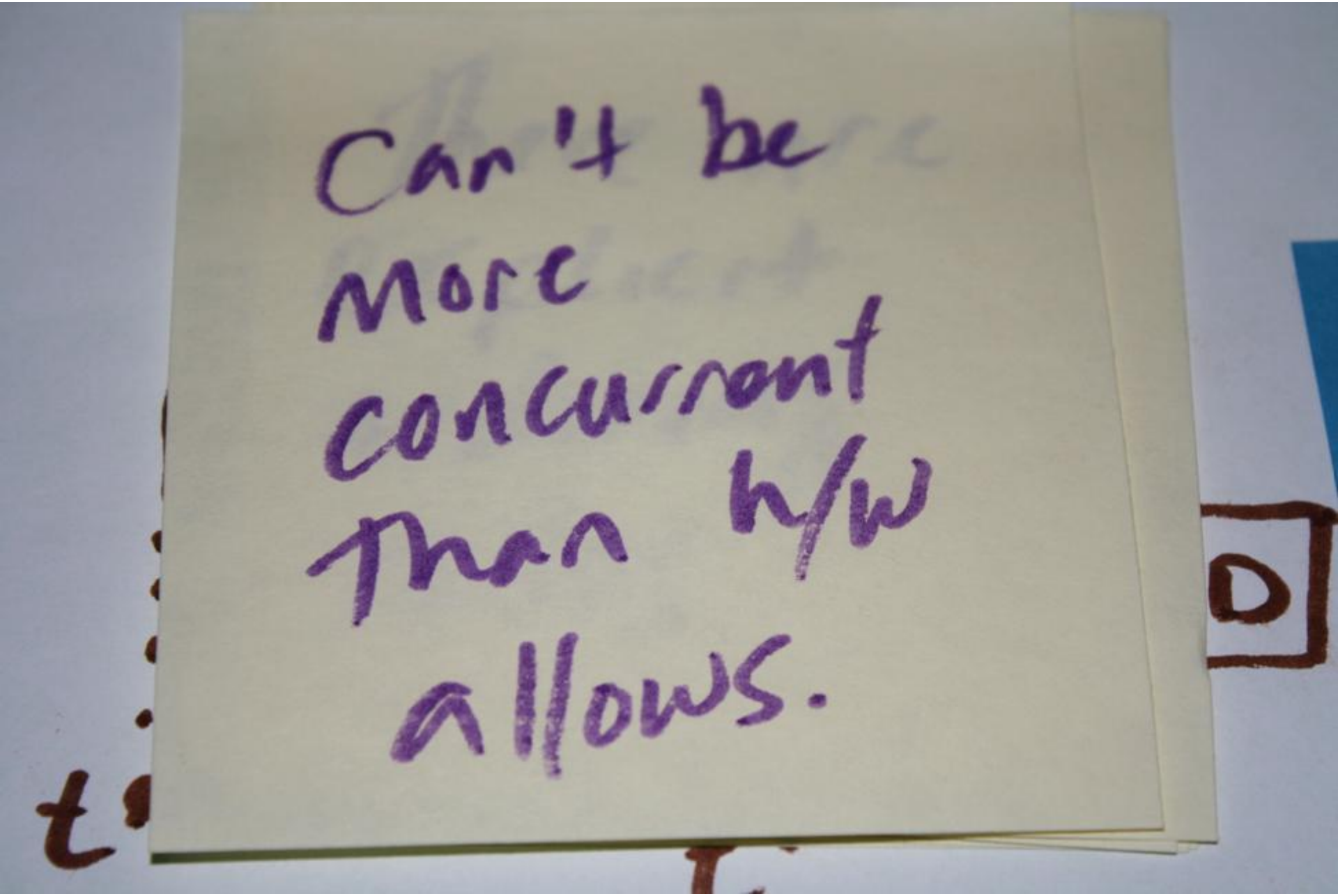
← JUST RUNNING SYNC APP ON >1 THREAD/CORE.

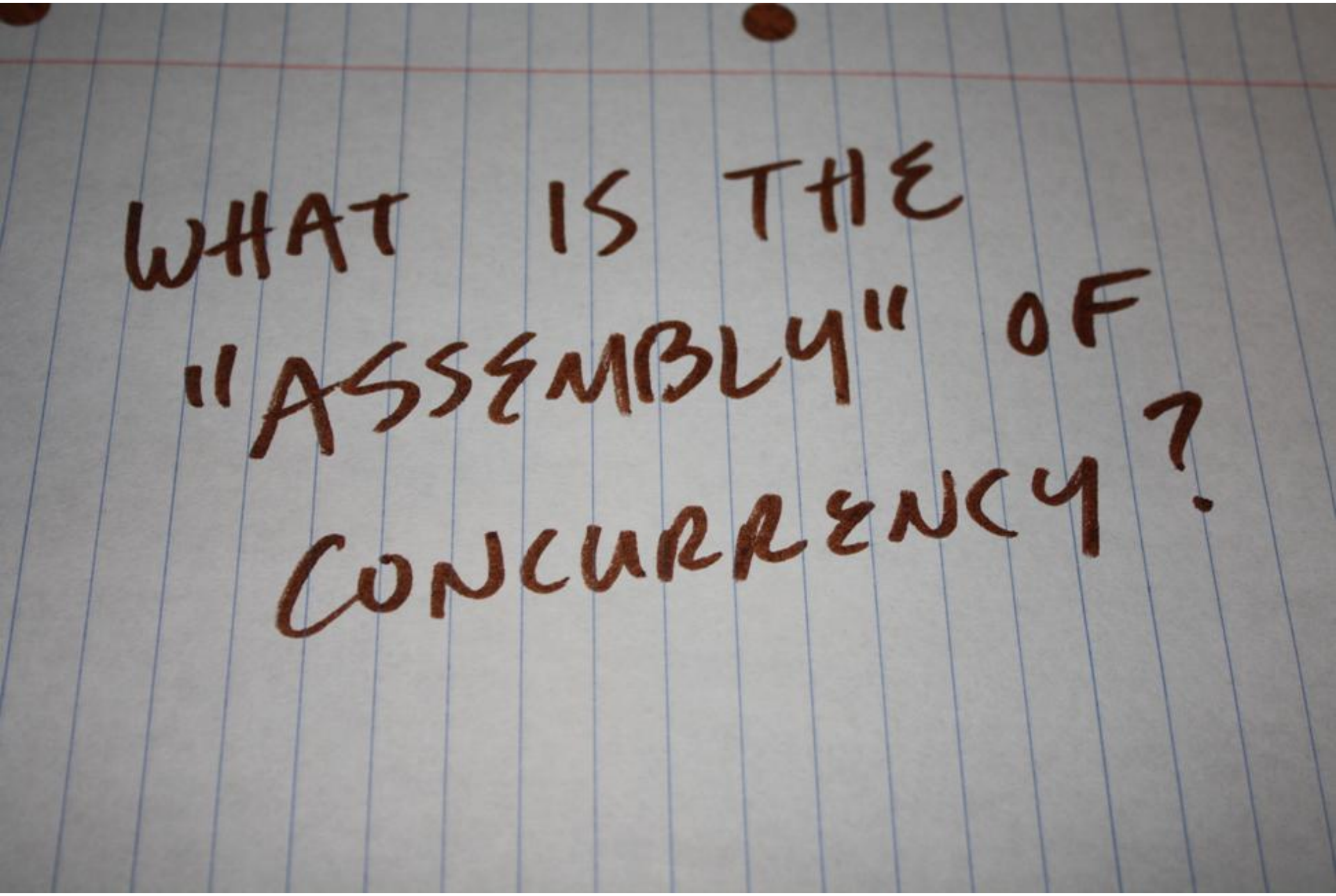

#2 THIS IS NOT CONCURRENCY!
However...
D
← JUST
SYNC


CONCURRENCY
CPUs
aren't
fully
concurrent
either
D

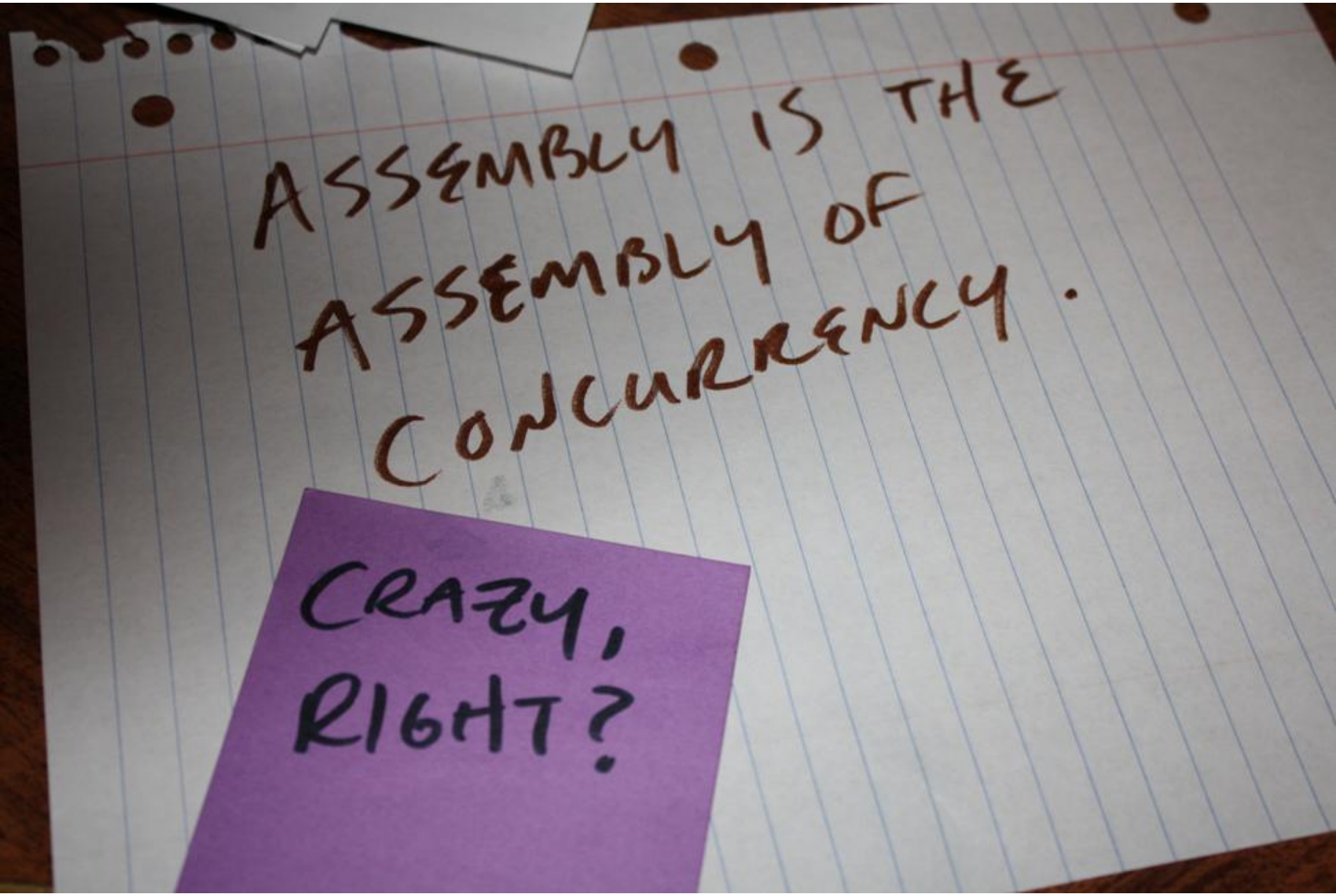

CONCURRENCY!
There is
a
clock
D

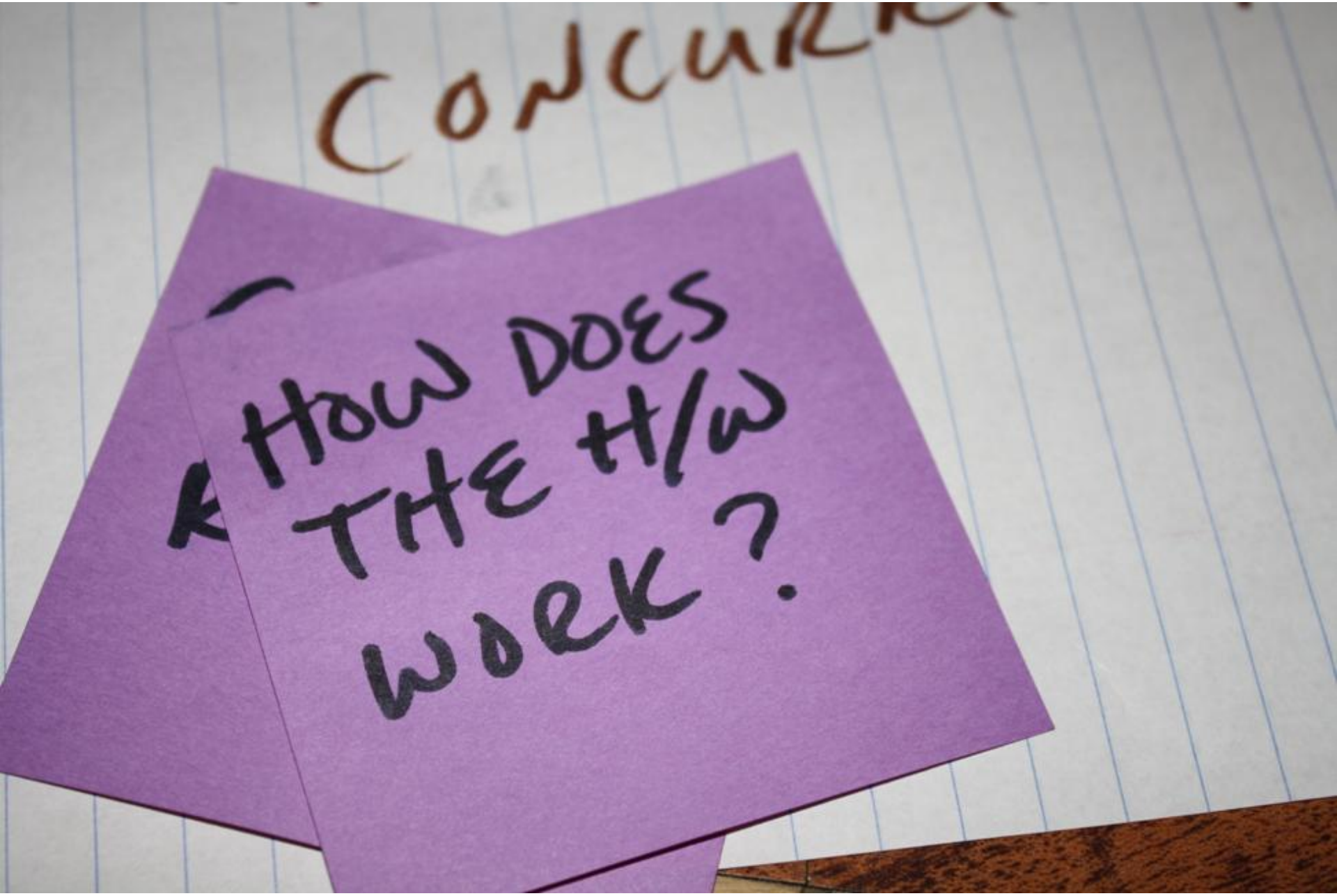

CONCURRENCY
There are explicit ordering rules
t
t


Can't be
more
concurrent
than h/w
allows.


WHAT IS THE
"ASSEMBLY" OF
CONCURRENCY?

ASSEMBLY IS THE ASSEMBLY OF CONCURRENCY.

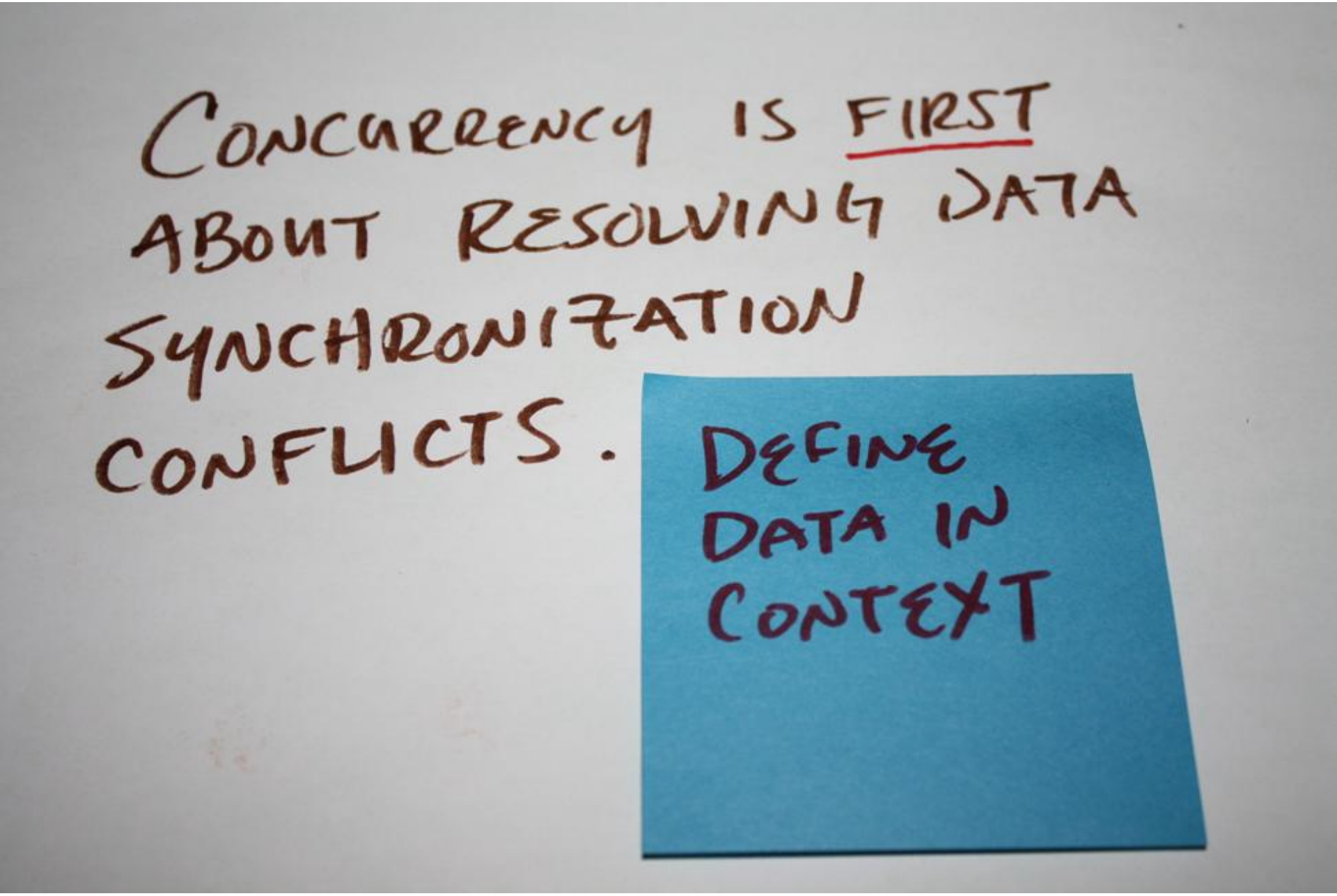

ASSEMBLY IS THE
ASSEMBLY OF
CONCURRENCY.
CRAZY,
RIGHT?

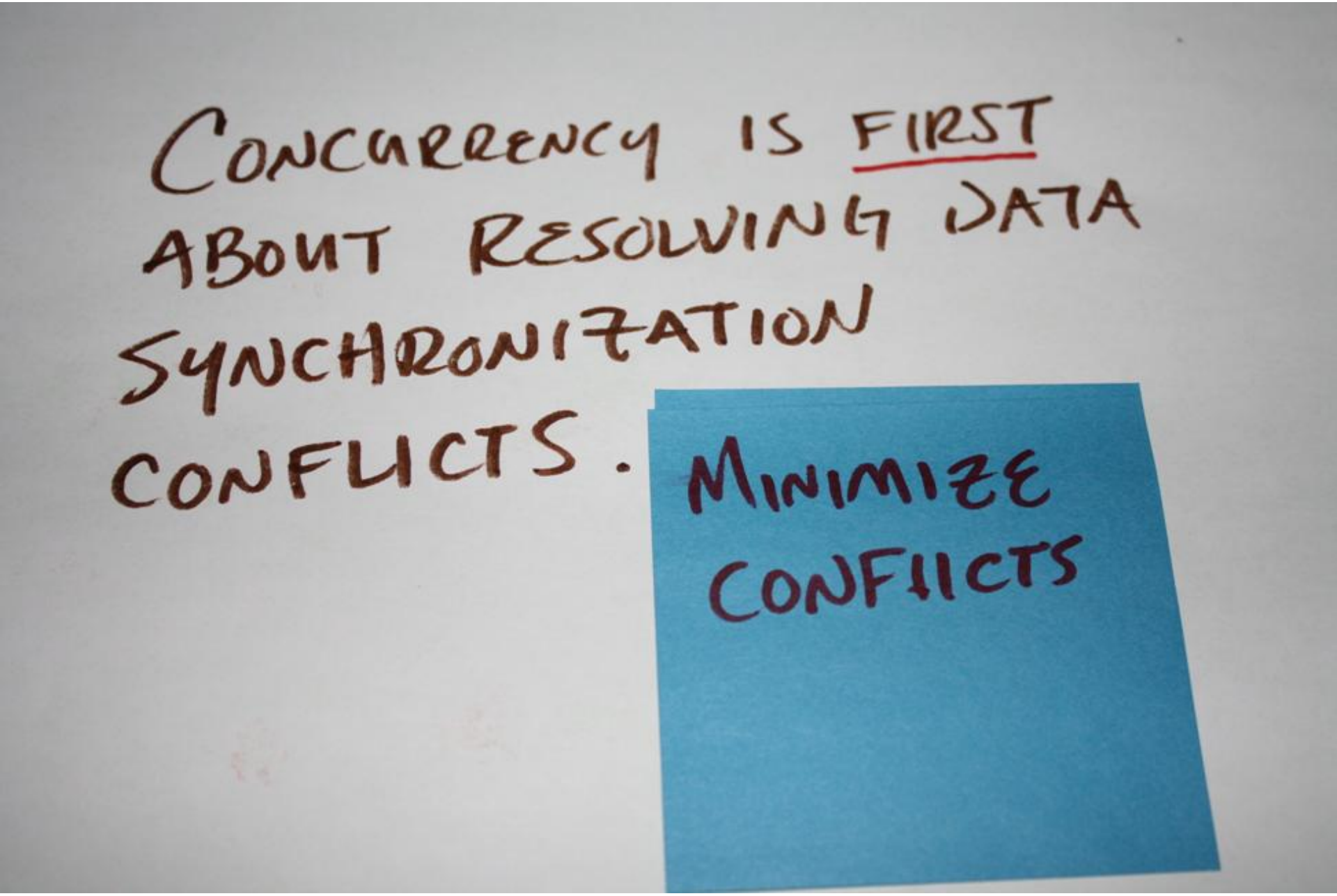

CONCURR
HOW DOES
THE H/W
WORK?

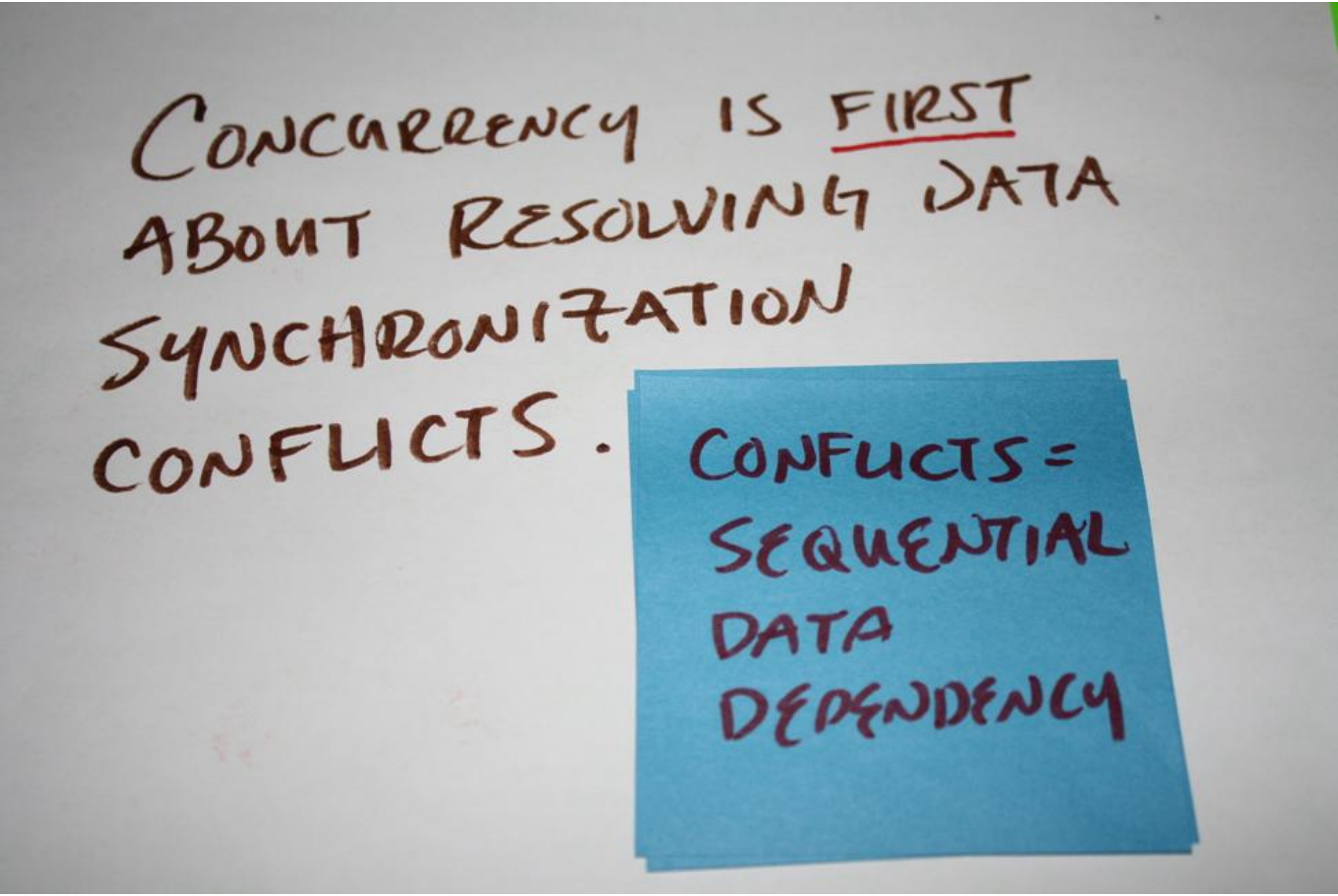
ATOMIC READ
ATOMIC WRITE

HOW DO THEY WORK
ON THE TARGET H/W?

CONCURRENCY IS FIRST ABOUT RESOLVING DATA SYNCHRONIZATION CONFLICTS.

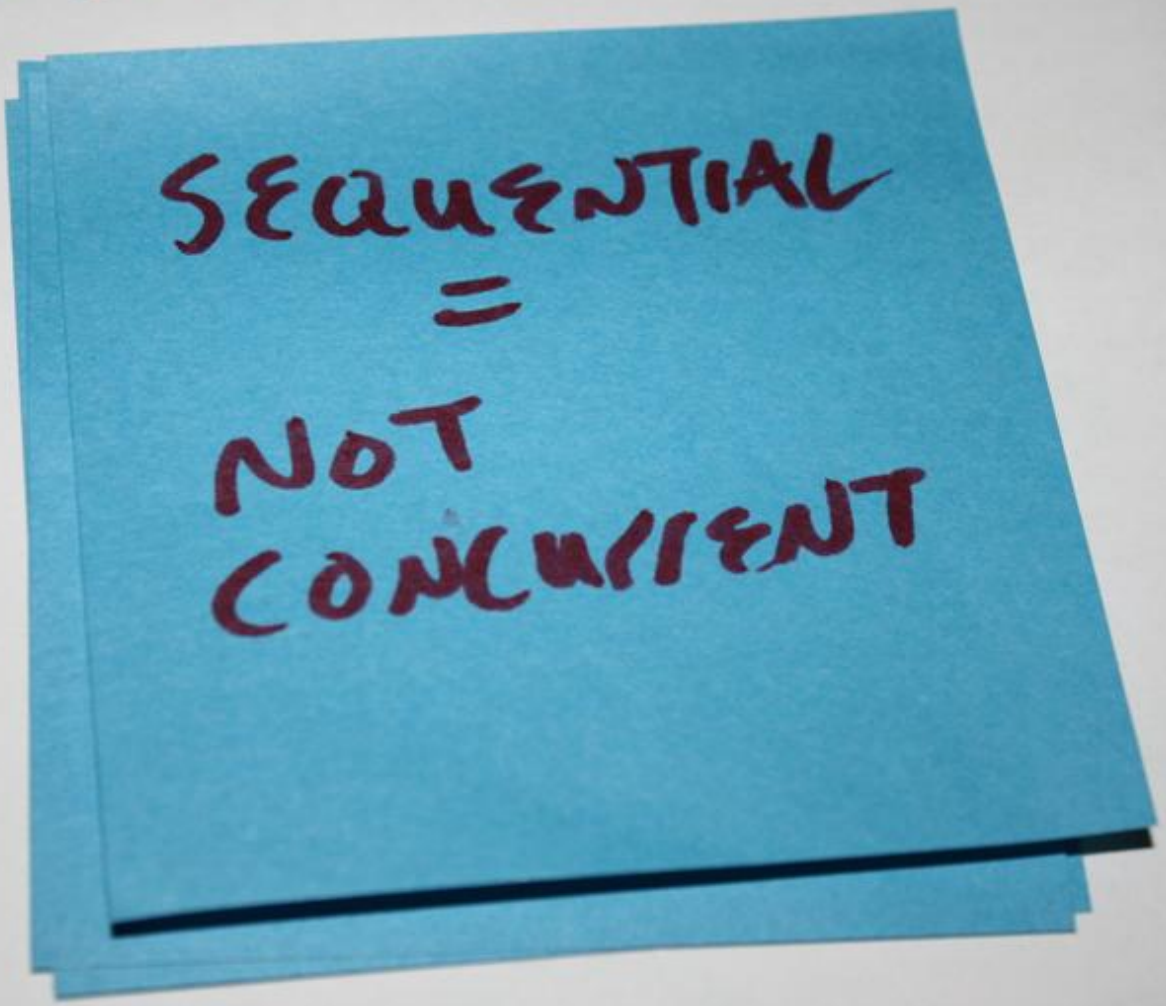

CONCURRENCY IS FIRST
ABOUT RESOLVING DATA
SYNCHRONIZATION
CONFLICTS.
DEFINE
DATA IN
CONTEXT


CONCURRENCY IS FIRST
ABOUT RESOLVING DATA
SYNCHRONIZATION
CONFLICTS.
MINIMIZE
CONFLICTS


CONCURRENCY IS FIRST
ABOUT RESOLVING DATA
SYNCHRONIZATION
CONFLICTS.
CONFLICTS =
SEQUENTIAL
DATA
DEPENDENCY

RONIZATION

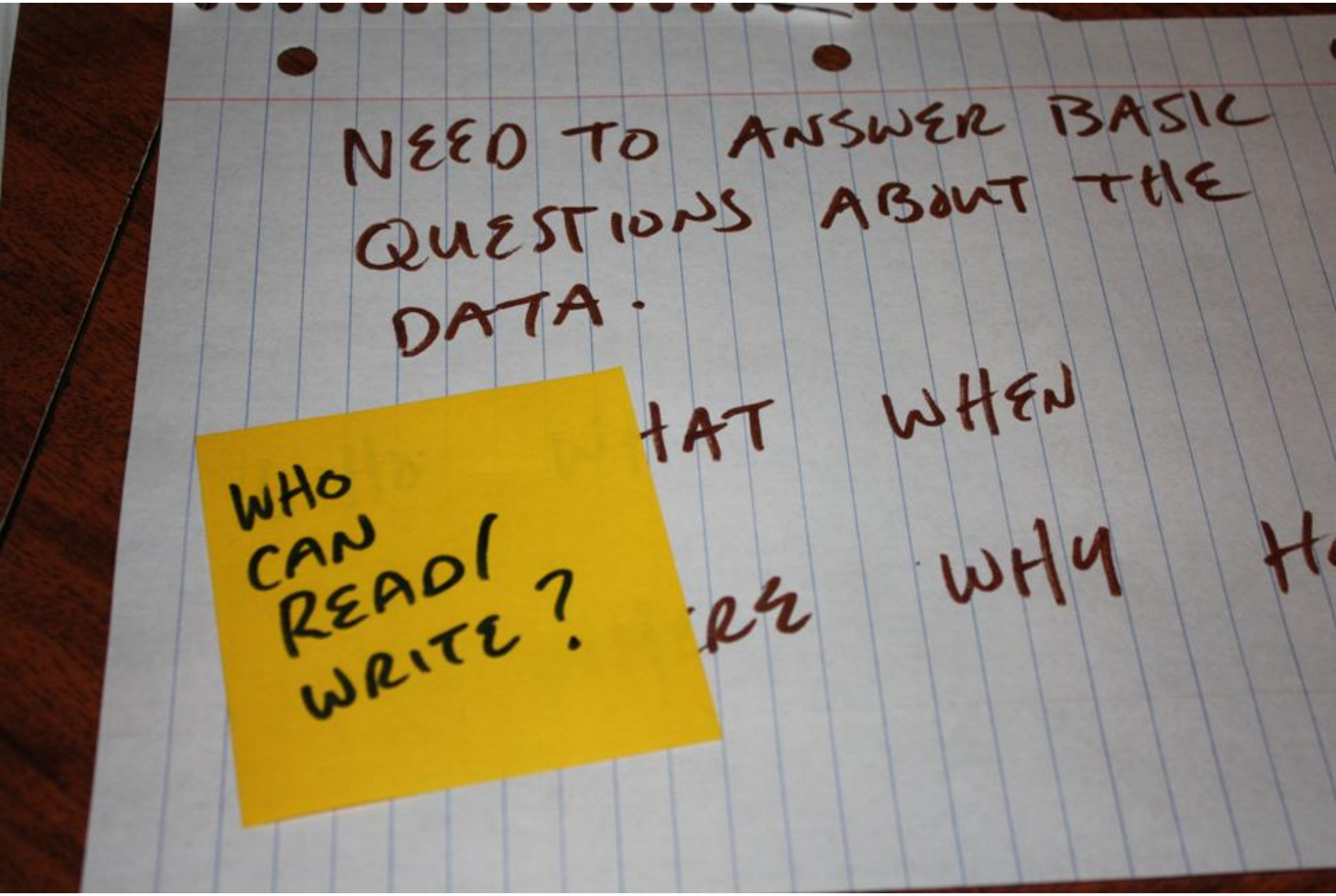
ICTS.

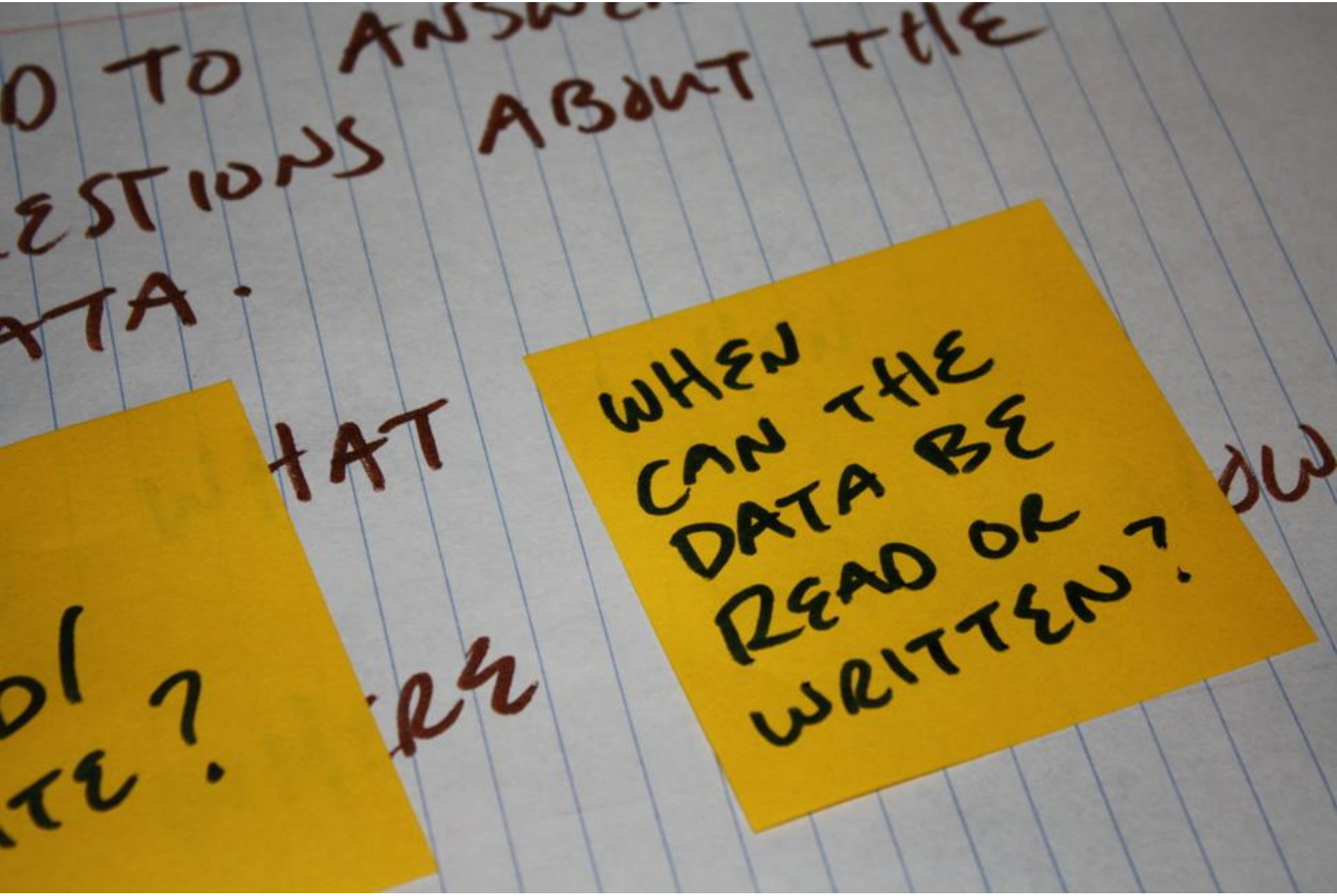


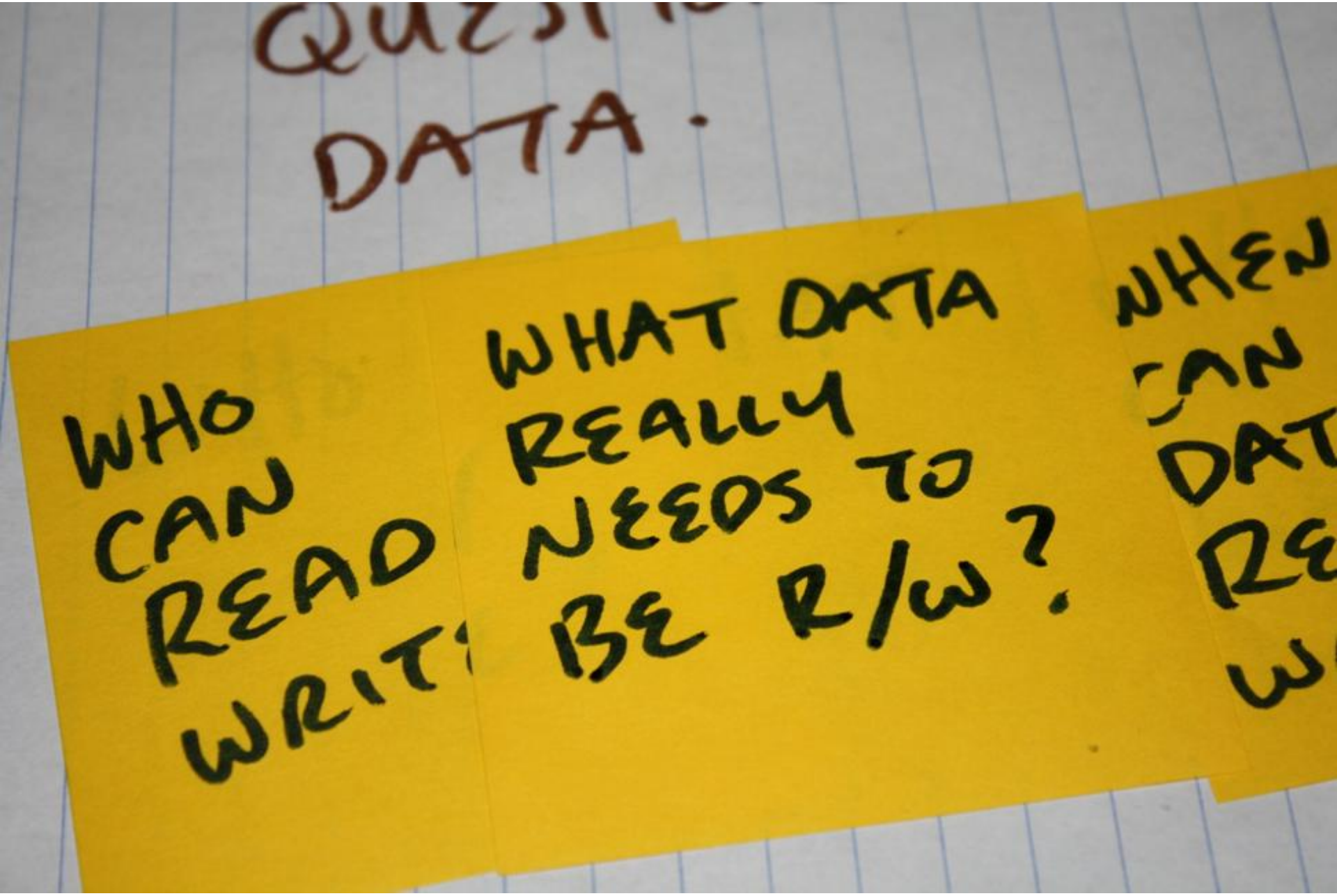
NEED TO ANSWER BASIC QUESTIONS ABOUT THE DATA.

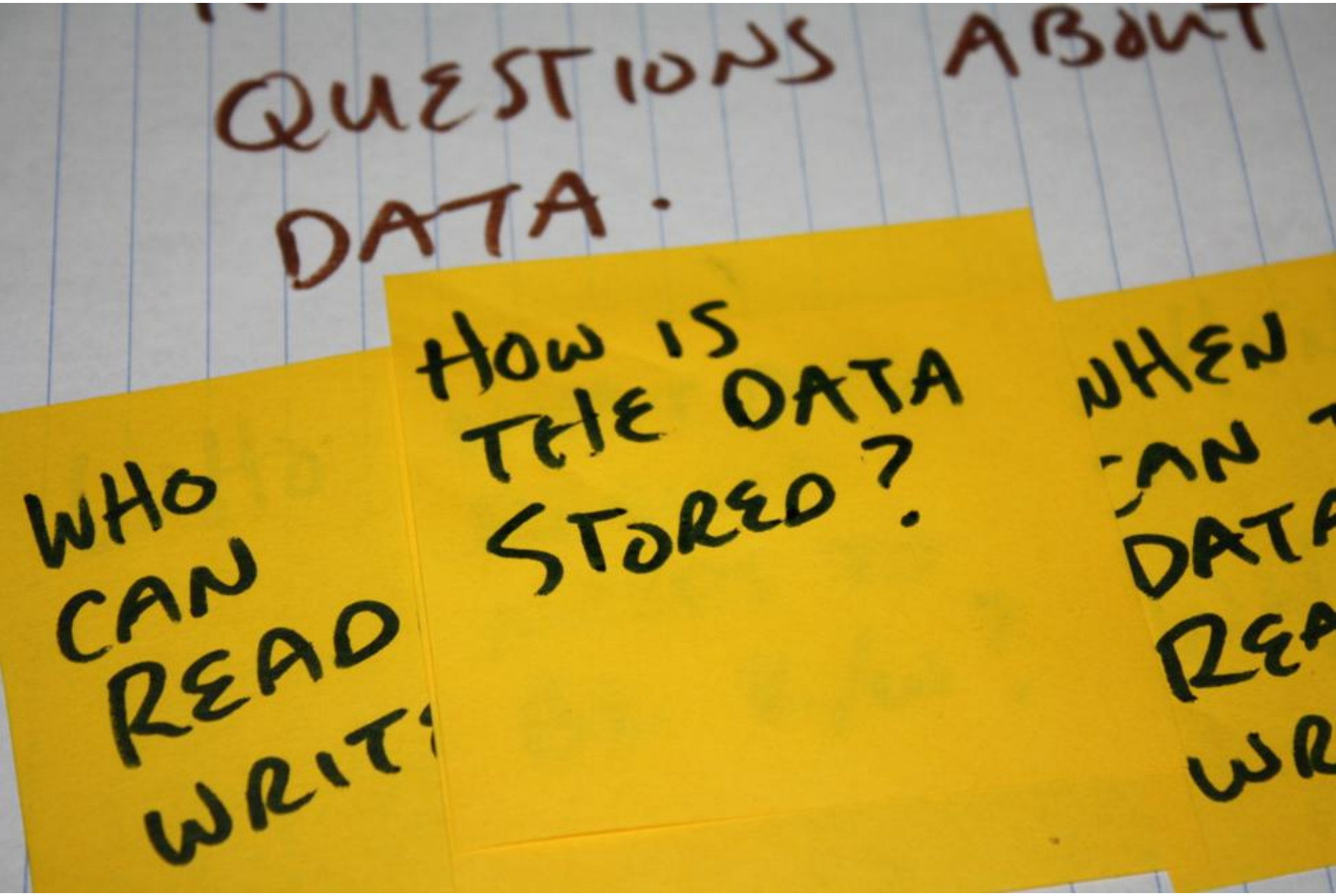
WHO WHAT WHEN

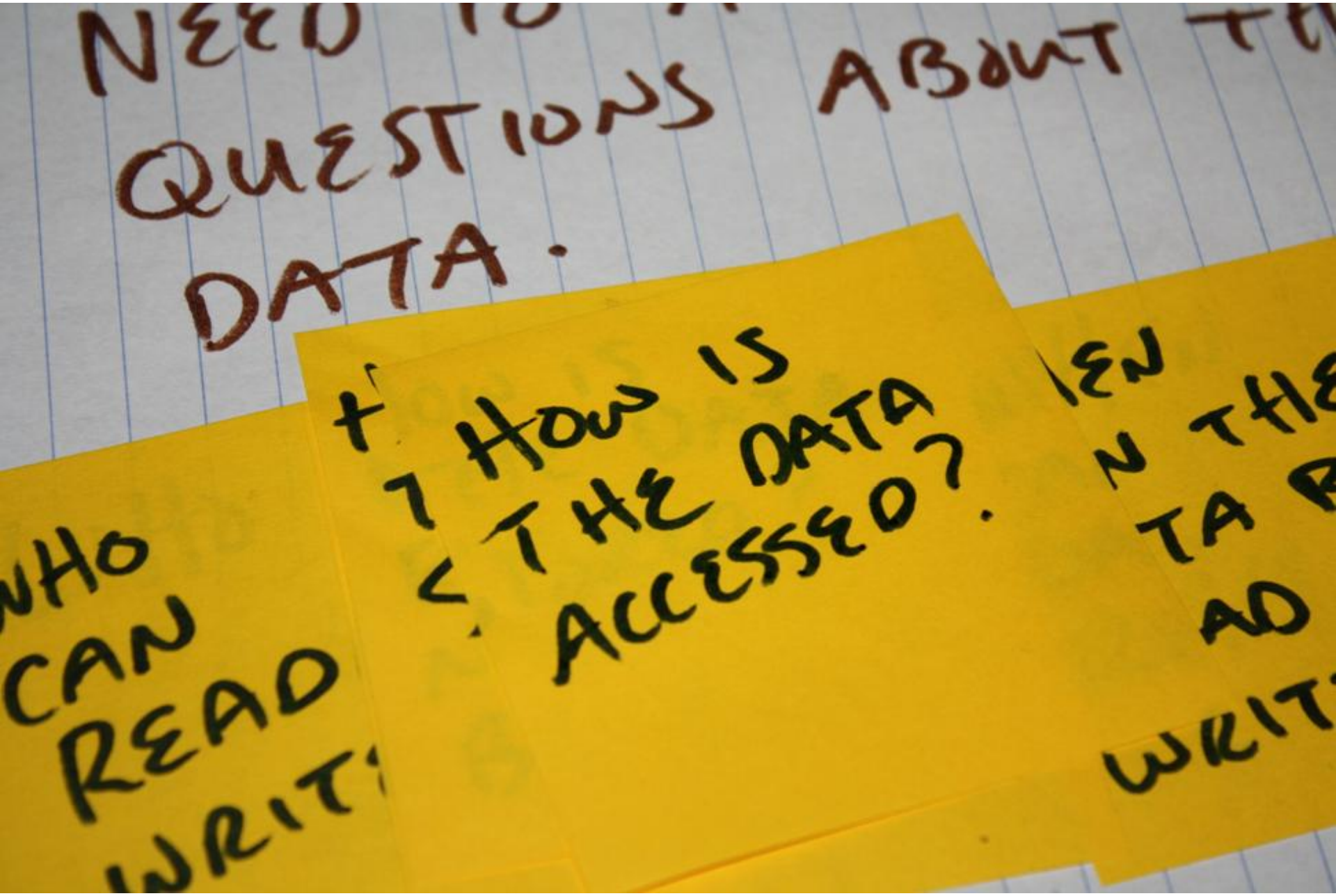
WHERE WHY HOW

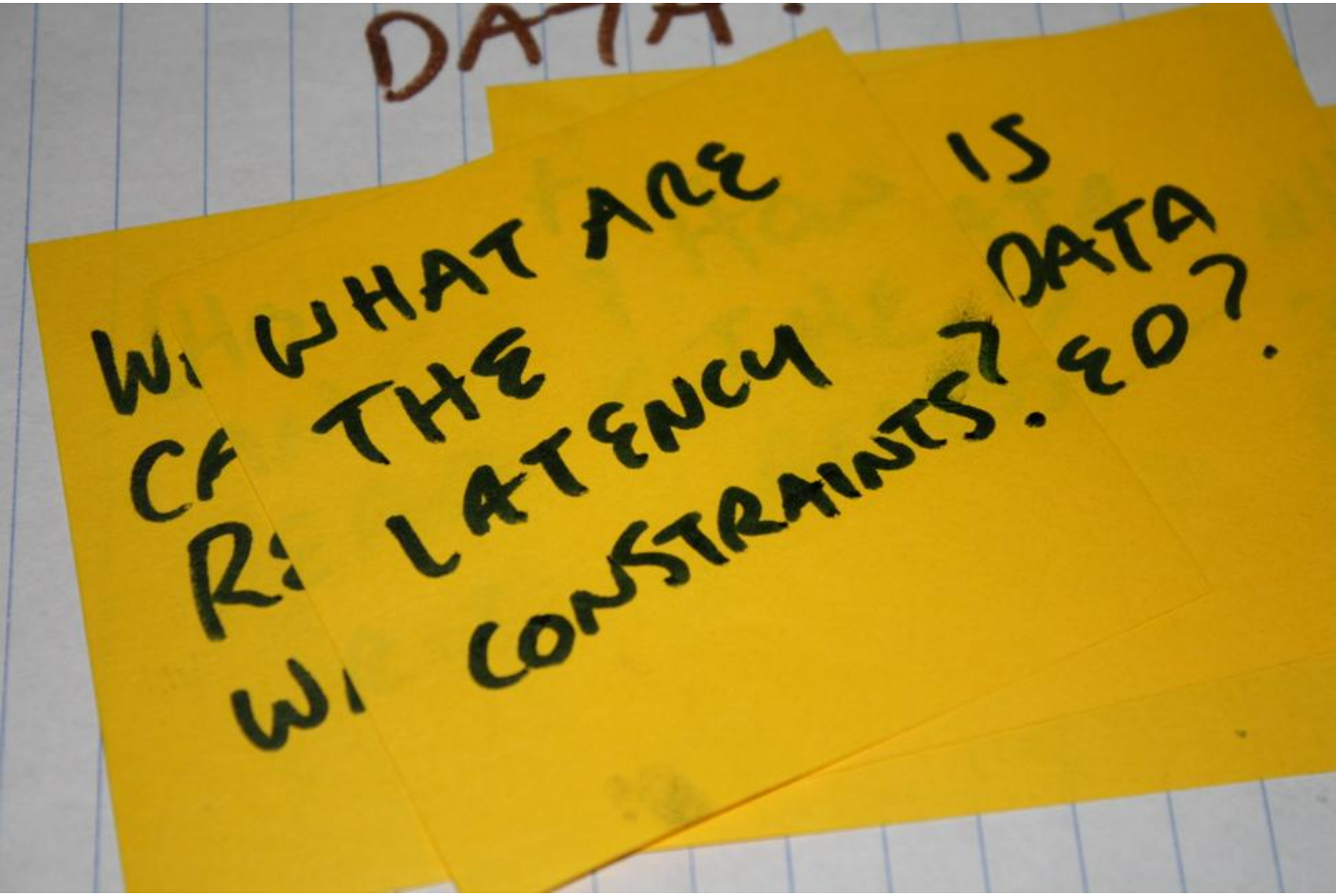

NEED TO ANSWER BASIC
QUESTIONS ABOUT THE
DATA.
HAT WHEN
WHO
CAN
READ/
WRITE?
RE WHY

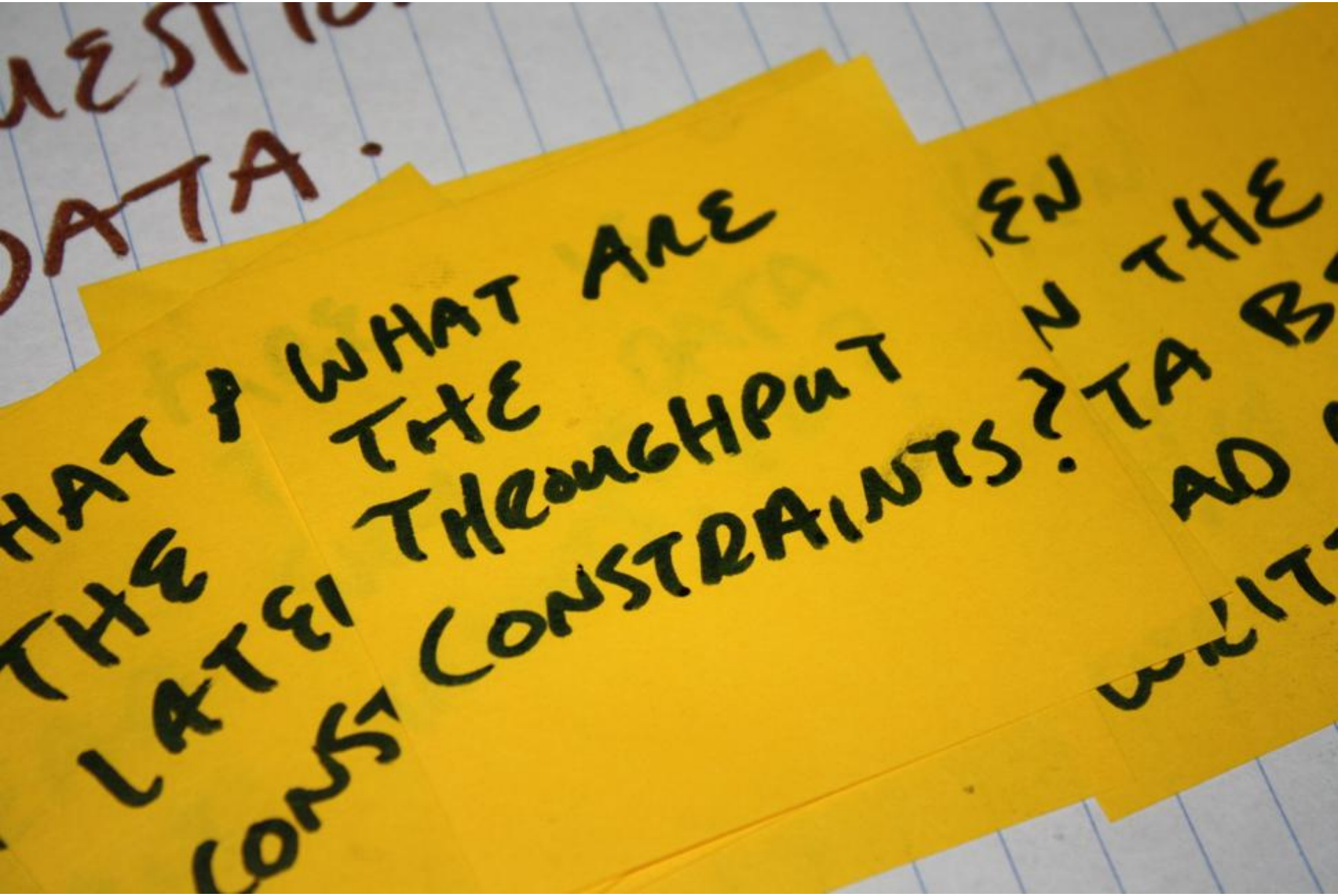

WHEN CAN THE DATA BE READ OR WRITTEN?

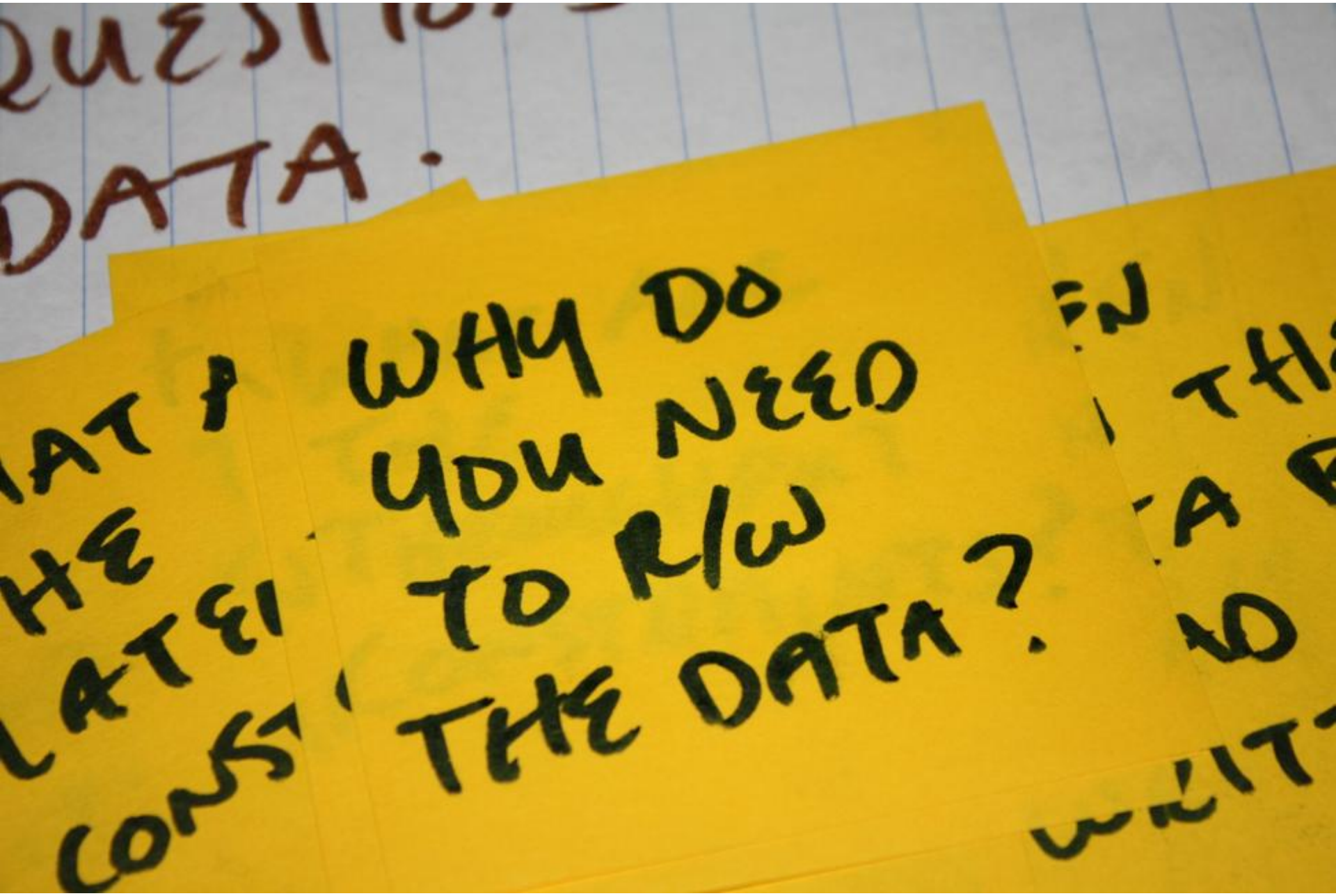

DATA.
WHO CAN READ
WHAT DATA REALLY NEEDS TO BE R/W?

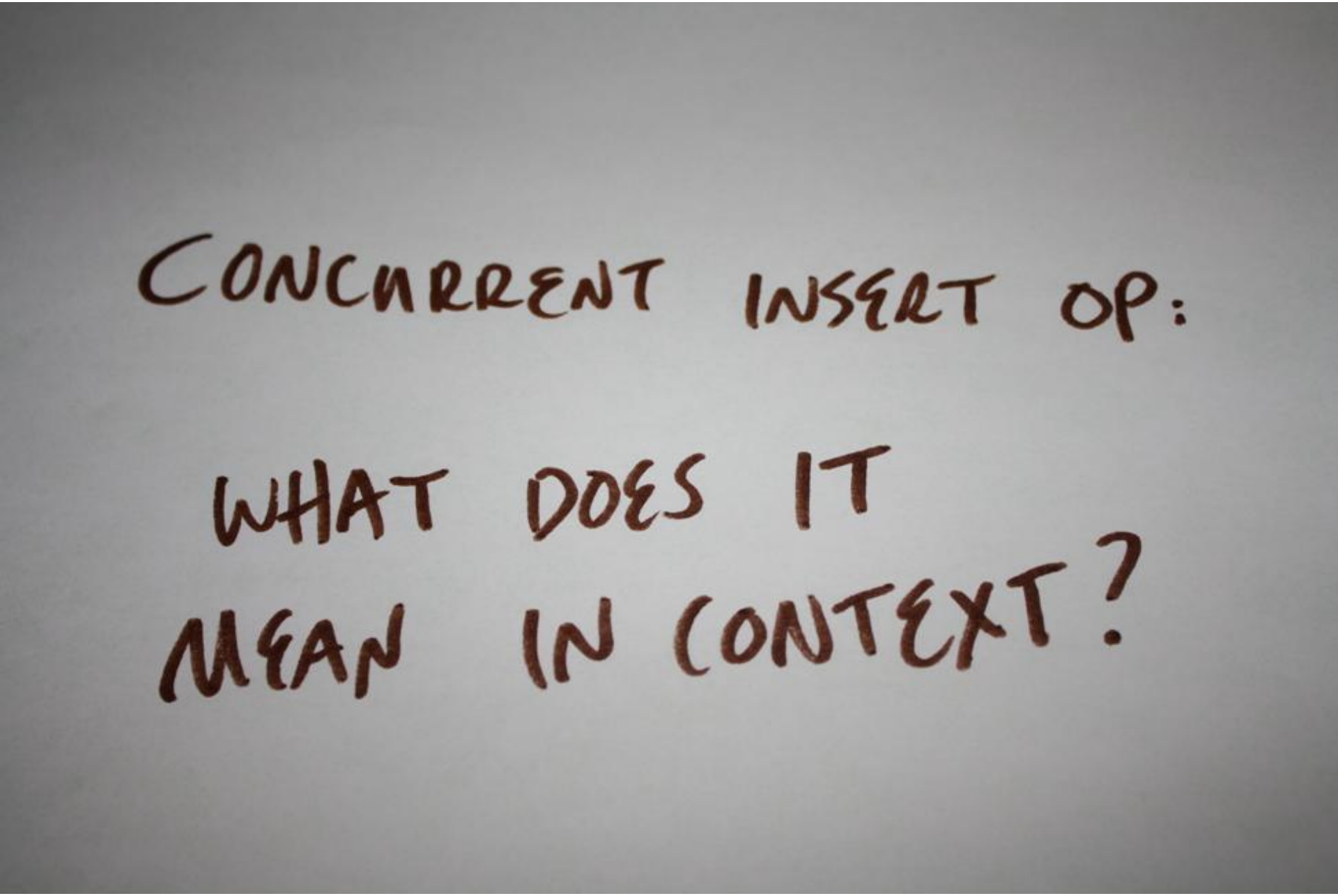

QUESTIONS ABOUT
DATA.
WHO
CAN
READ
WRIT
HOW IS
THE DATA
STORED?
WHEN
AN
DATA
REA
WR


NEED
QUESTIONS ABOUT
DATA.
WHO
CAN
READ
HOW IS
THE DATA
ACCESSED?
WHEN
IN THE
DATA
ADD
WRITE


DATA
WHAT ARE
THE
LATENCY
CONSTRAINTS?
IS
DATA
GO?


WHAT ARE
THE
THROUGHPUT
CONSTRAINTS?

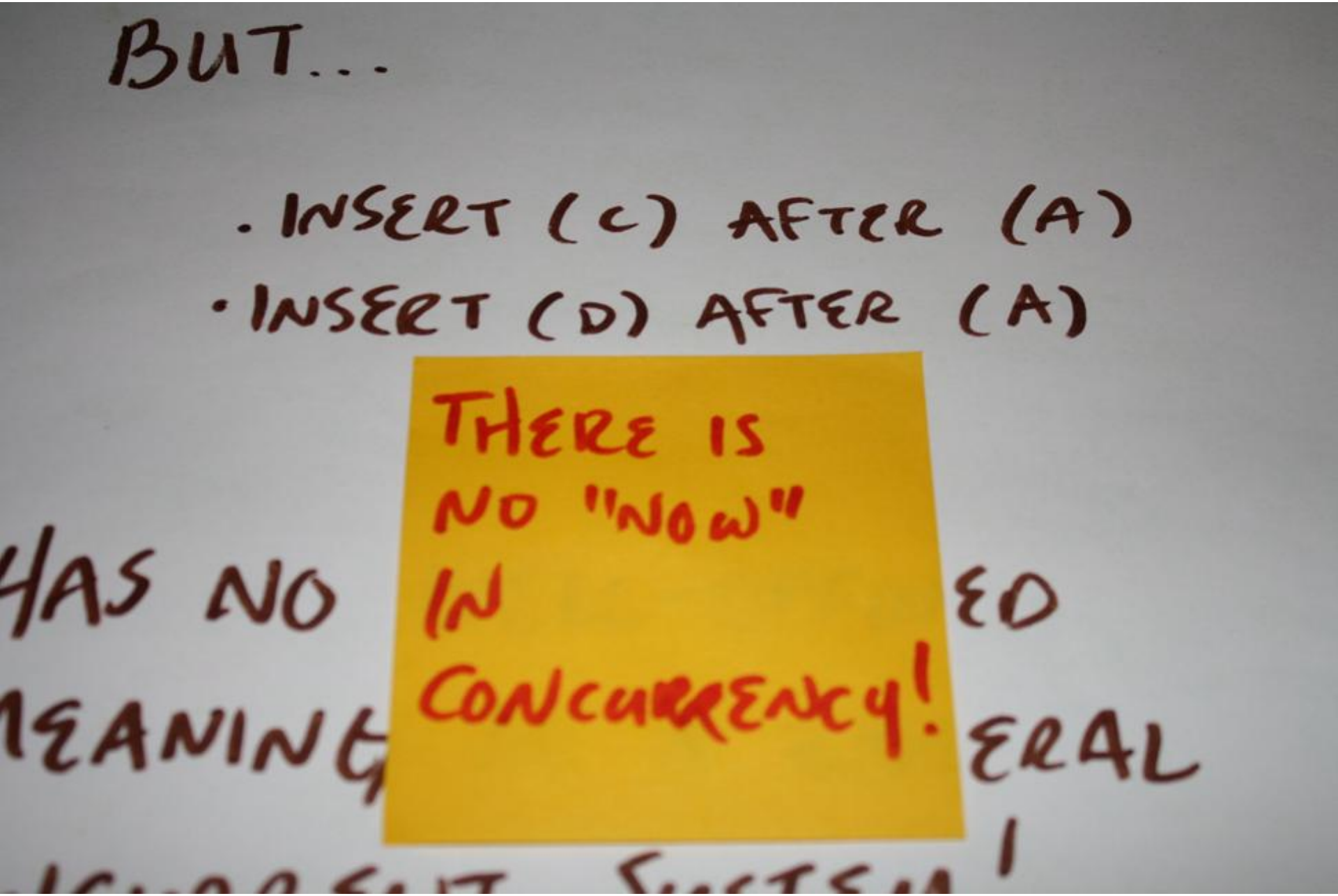

WHY DO
YOU NEED
TO R/W
THE DATA?

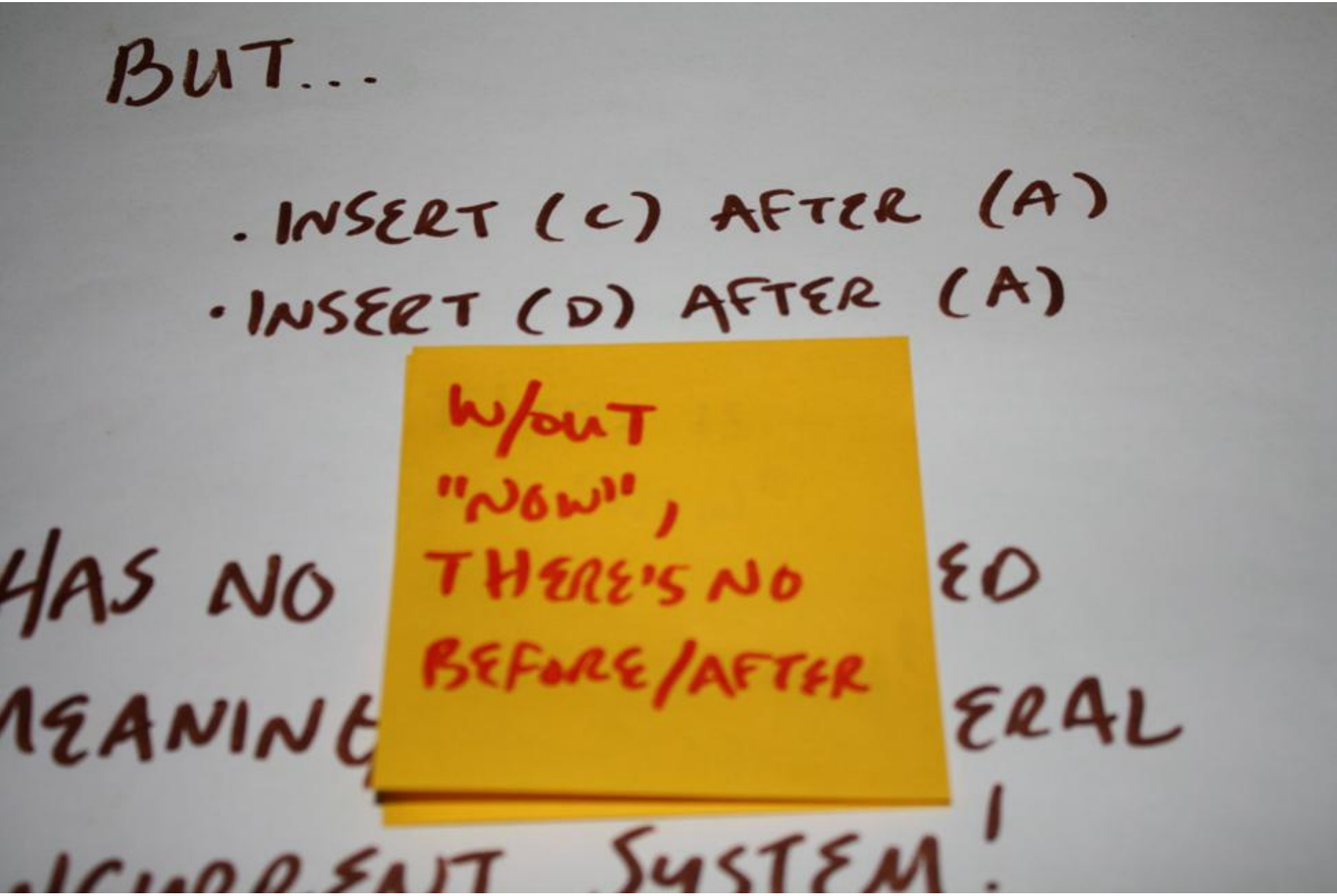

CONCURRENT INSERT OP:
WHAT DOES IT
MEAN IN CONTEXT?

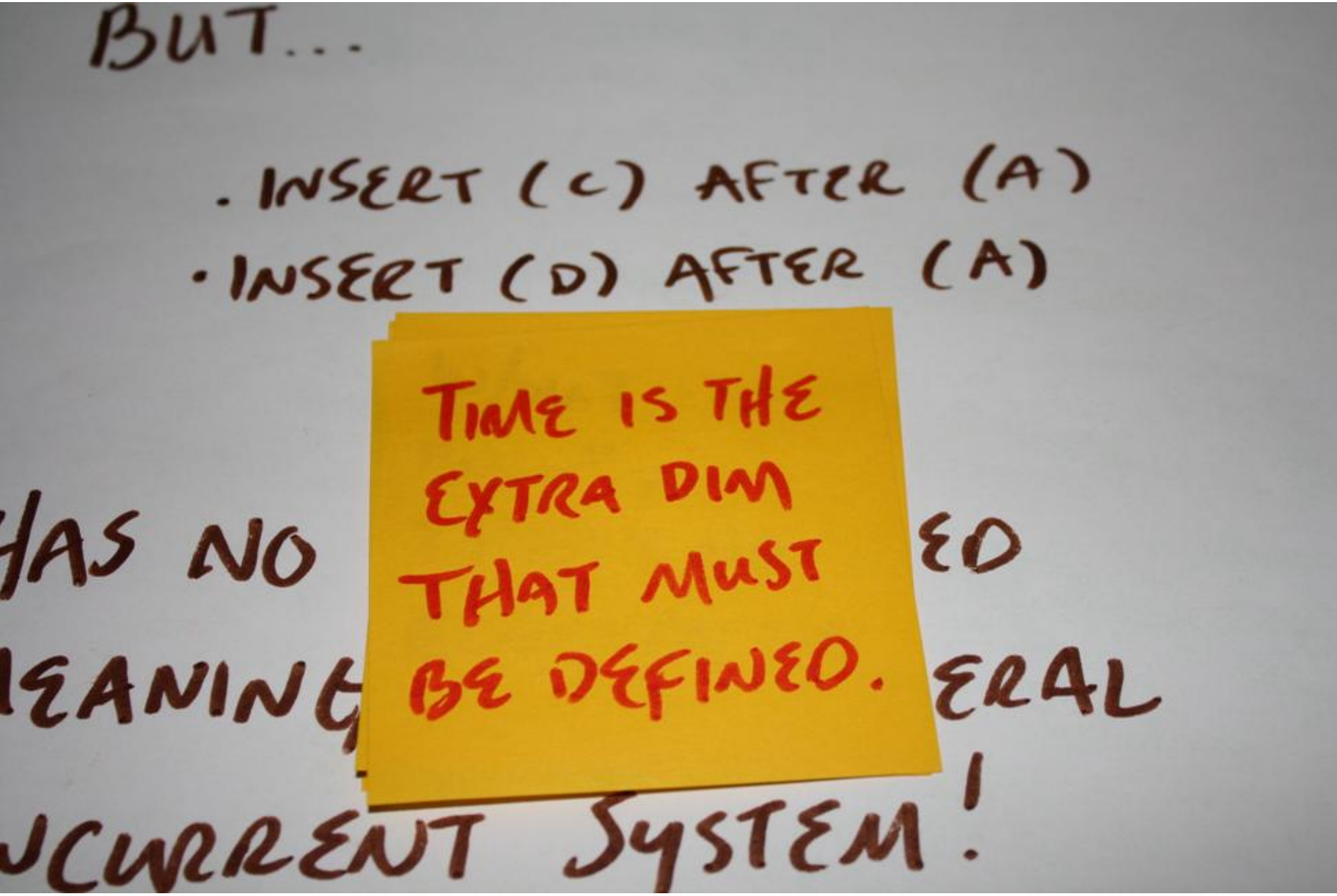
BUT...

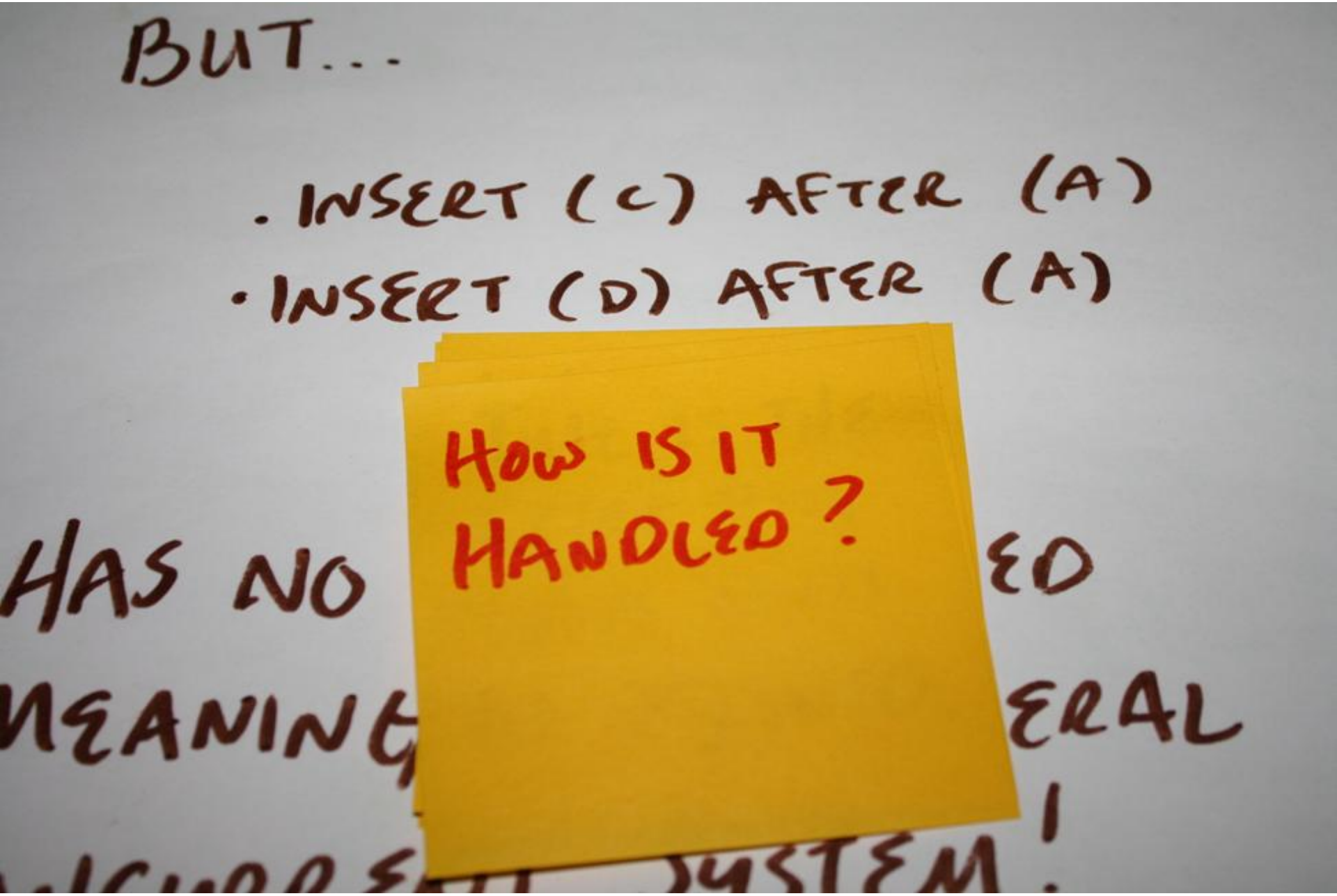
- INSERT (C) AFTER (A)
- INSERT (D) AFTER (A)

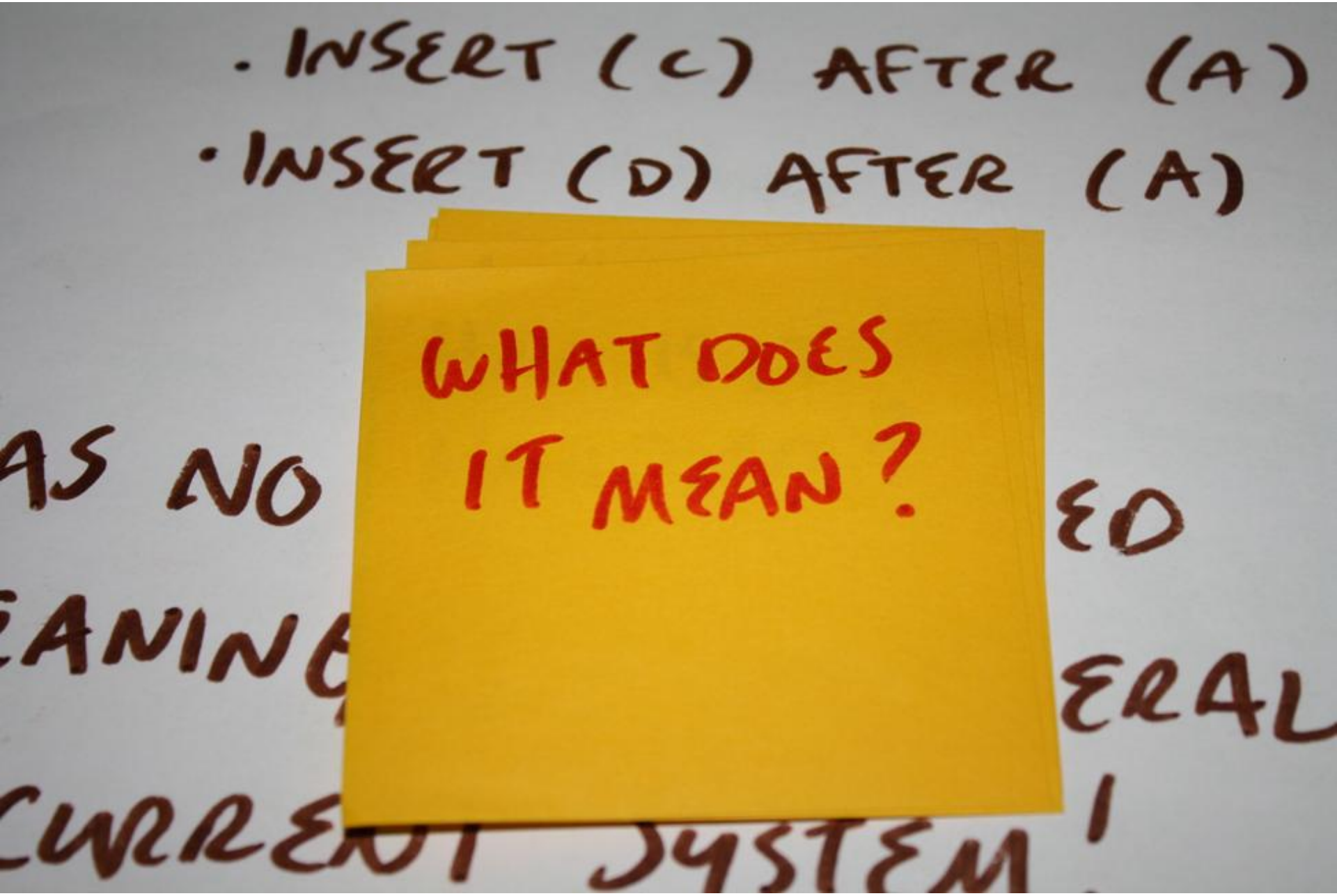
HAS NO WELL-DEFINED MEANING IN A GENERAL CONCURRENT SYSTEM!

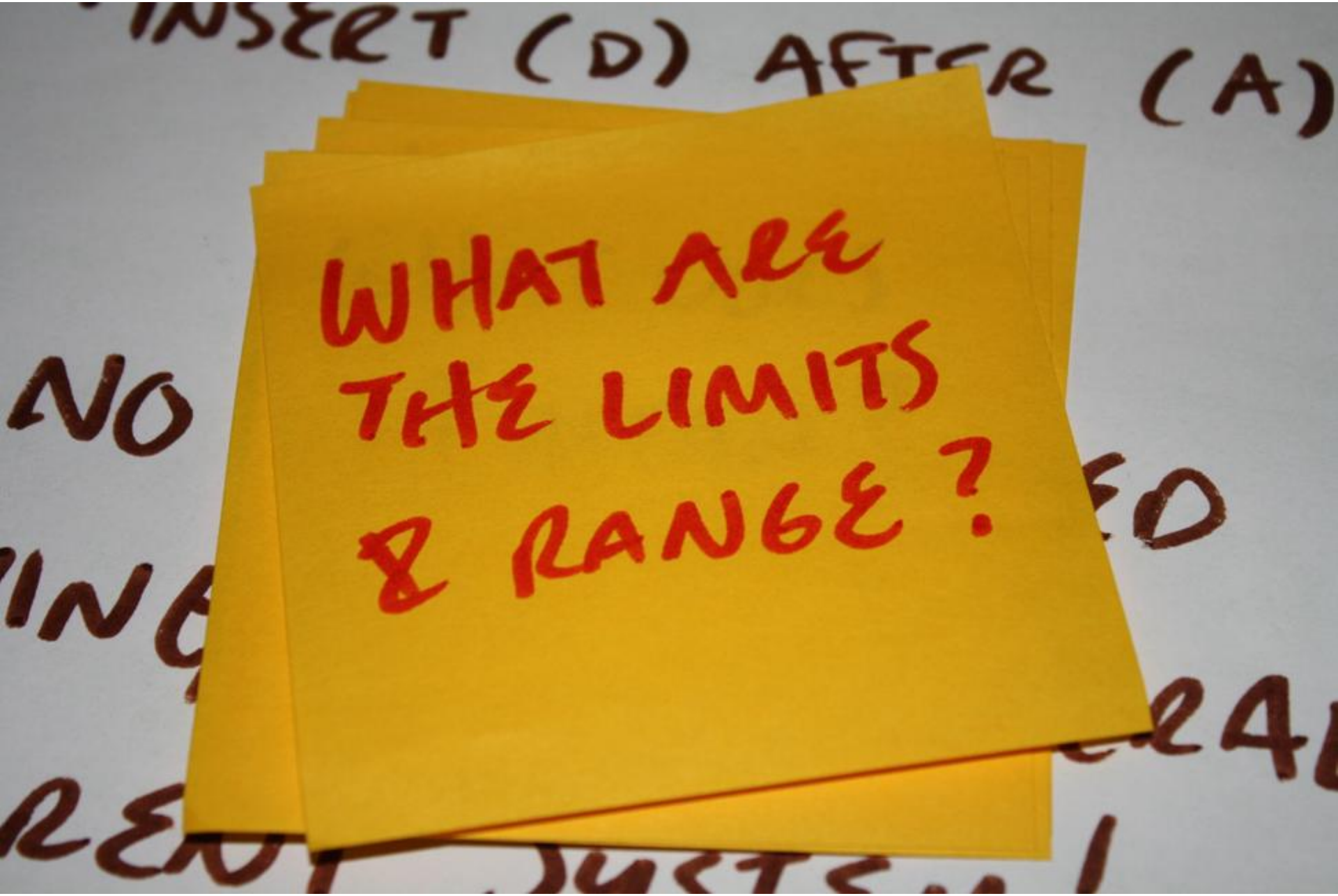

BUT...
. INSERT (C) AFTER (A)
· INSERT (D) AFTER (A)
THERE IS
NO "NOW"
IN
CONCURRENCY!


BUT...
. INSERT (C) AFTER (A)
• INSERT (D) AFTER (A)
W/OUT "NOW", THERE'S NO BEFORE/AFTER
HAS NO
ED
MEANIN
ERAL
SYSTEM!

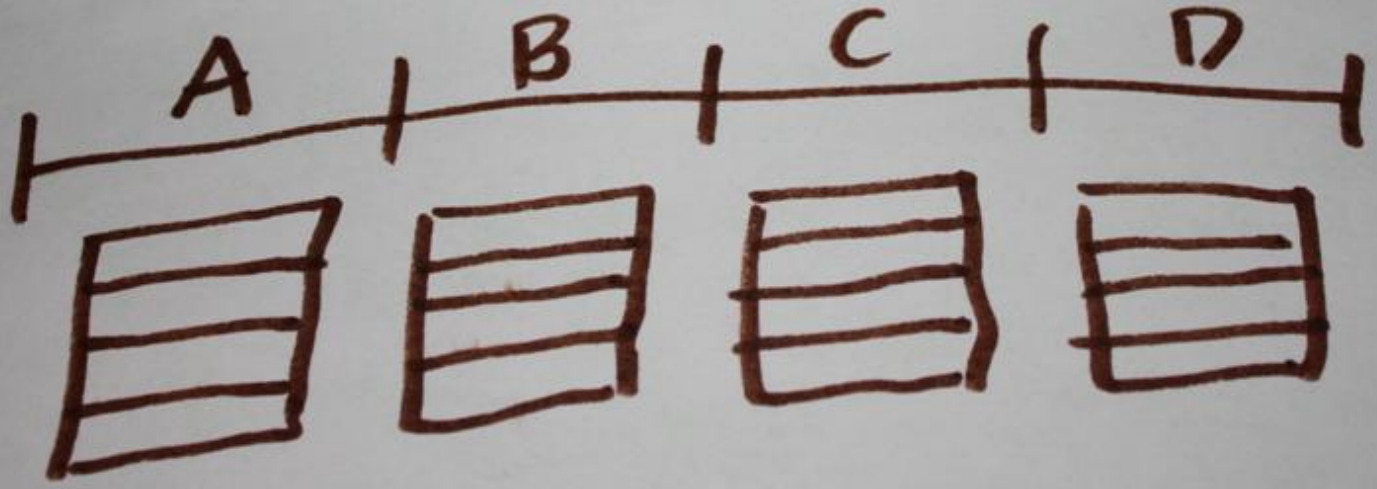

BUT...
. INSERT (C) AFTER (A)
· INSERT (D) AFTER (A)
TIME IS THE EXTRA DIM THAT MUST BE DEFINED.
HAS NO
ED
EANING
ERAL
NCURRENT SYSTEM!


BUT...
. INSERT (C) AFTER (A)
· INSERT (D) AFTER (A)
HOW IS IT HANDLED?
HAS NO
ED
MEANIN
ERAL
SYSTEM!

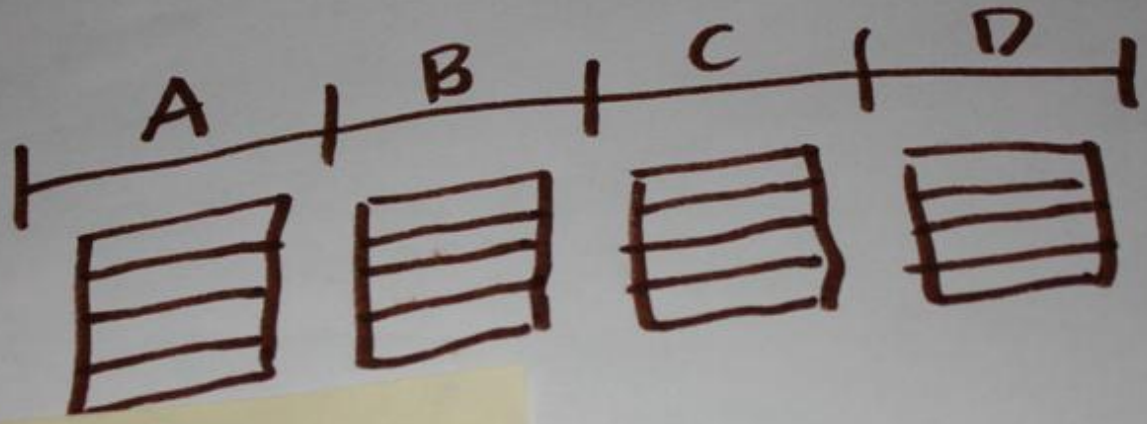

. INSERT (C) AFTER (A)
• INSERT (D) AFTER (A)
WHAT DOES IT MEAN?


INSERT (D) AFTER (A)
WHAT ARE
THE LIMITS
& RANGE?

WHAT IS THE ACCURACY / GRANULARITY OF GLOBAL ORDER VALUES?

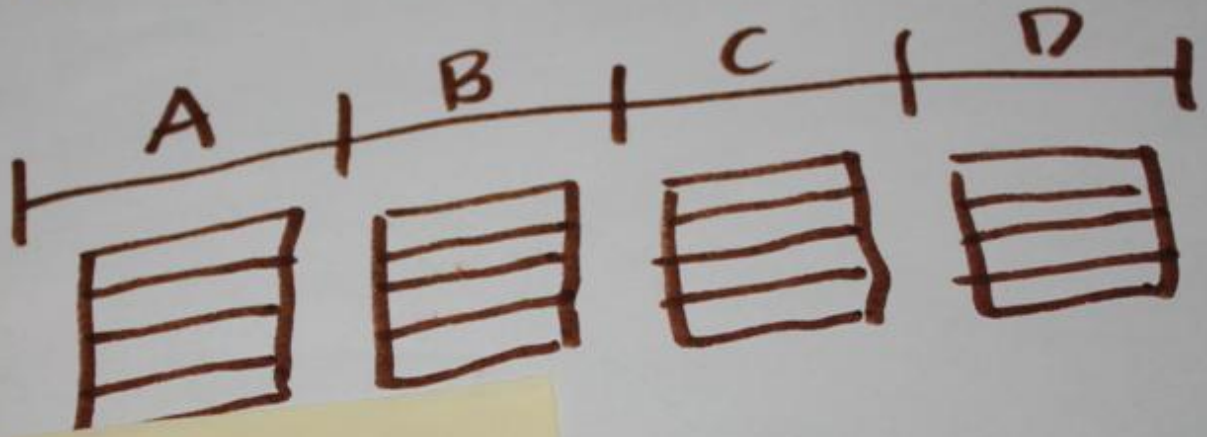


GRANULARITY
ORDER VALUES?

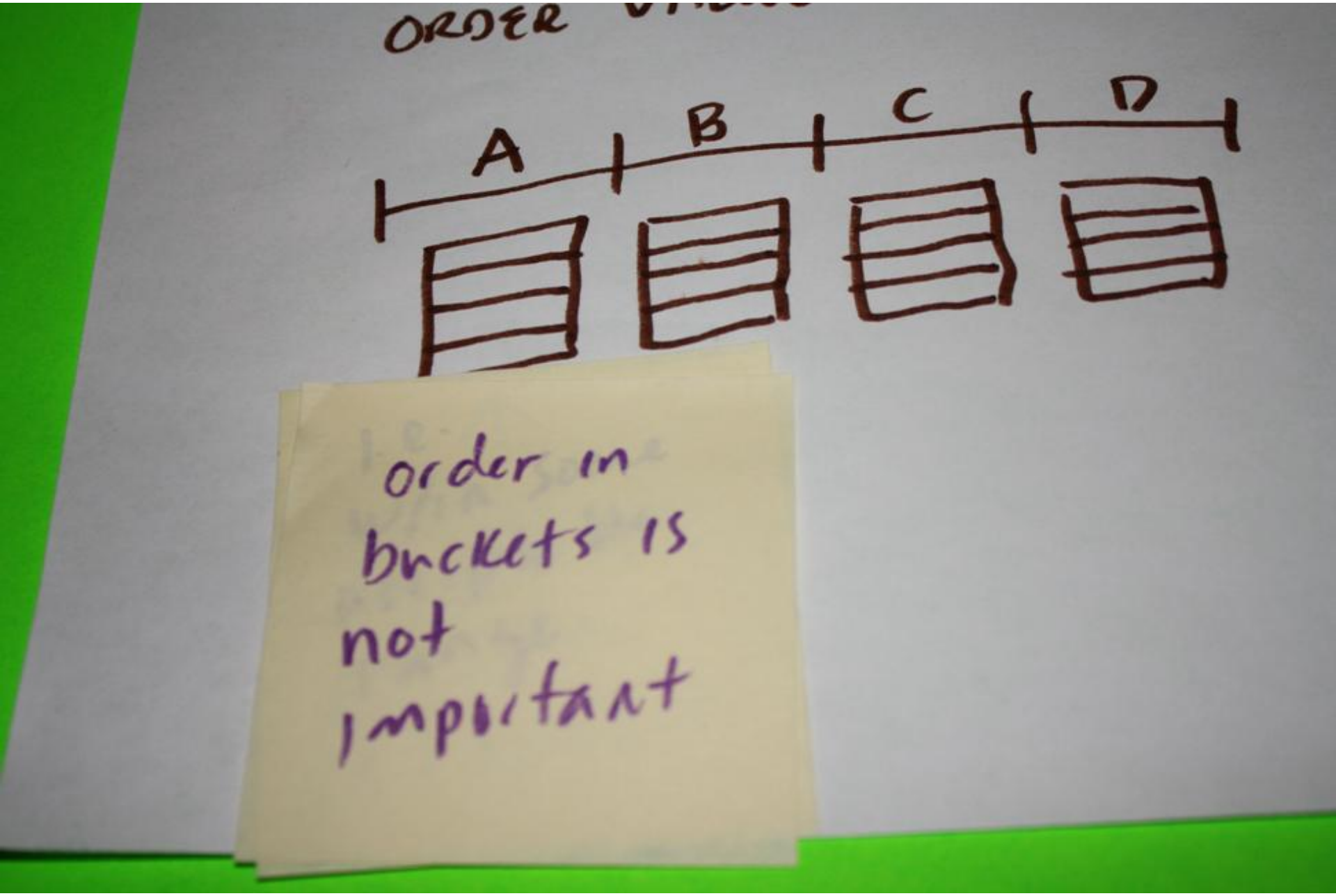


Maybe "Insert A" means in (A) bucket.

ORDER VALUES.

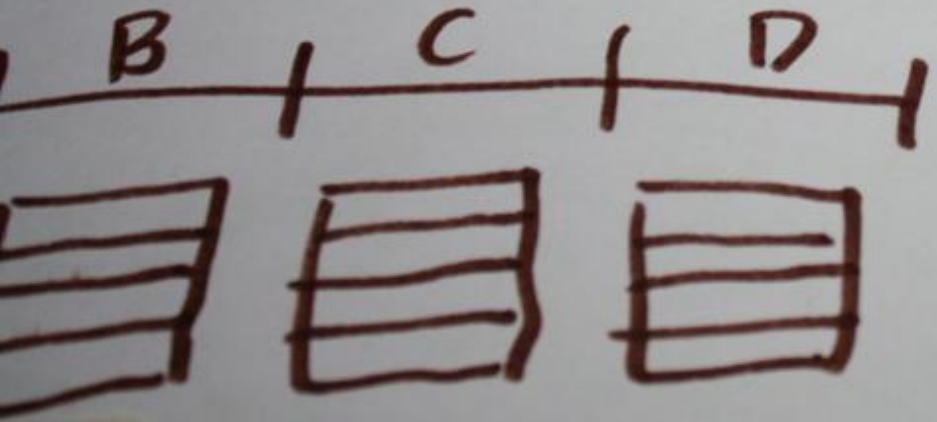


i.e.
w/in some
acceptable
range.


ORDER
A
B
C
D
order in
buckets is
not
important

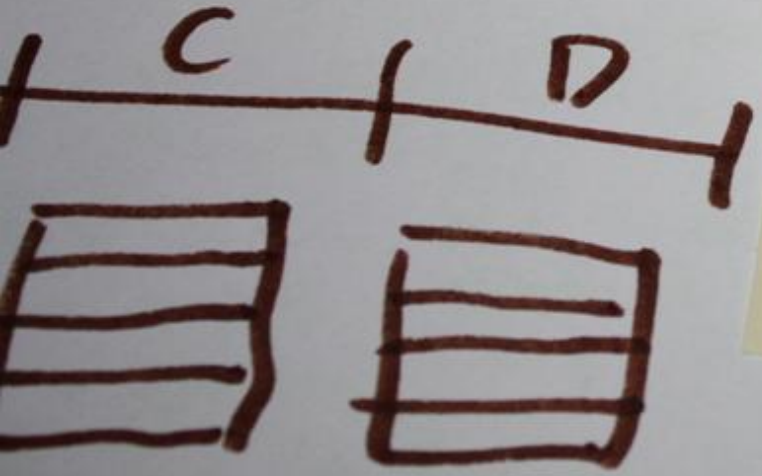

ORDER
A
B
C
D
but global
order is
well-defined.

IS THE ACCURACY /
ULARITY OF GLOBAL
VALUES?

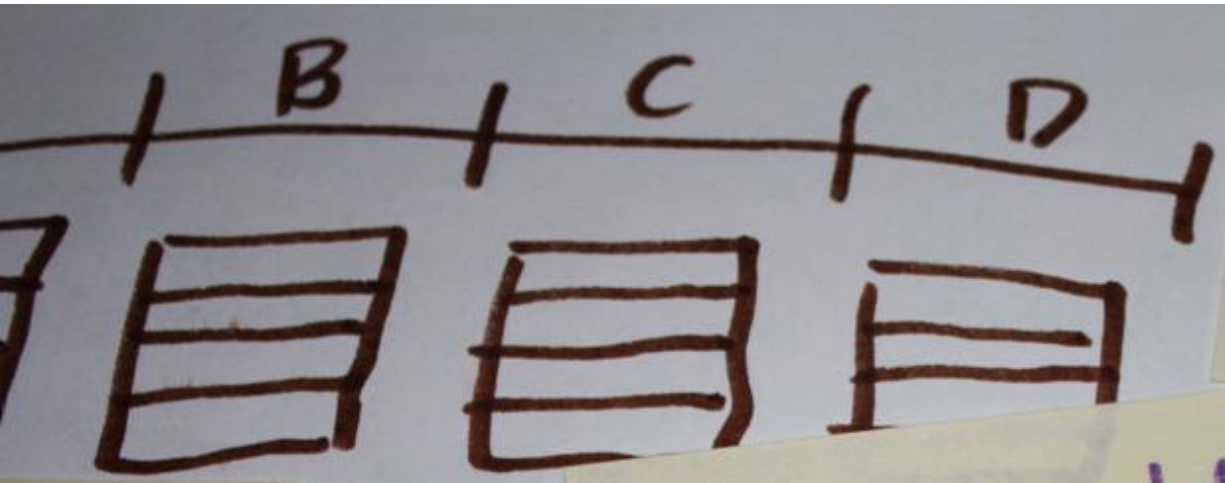


This is an example of an ordering rule.

THE ACCURACY /
ITY OF GLOBAL
UES?



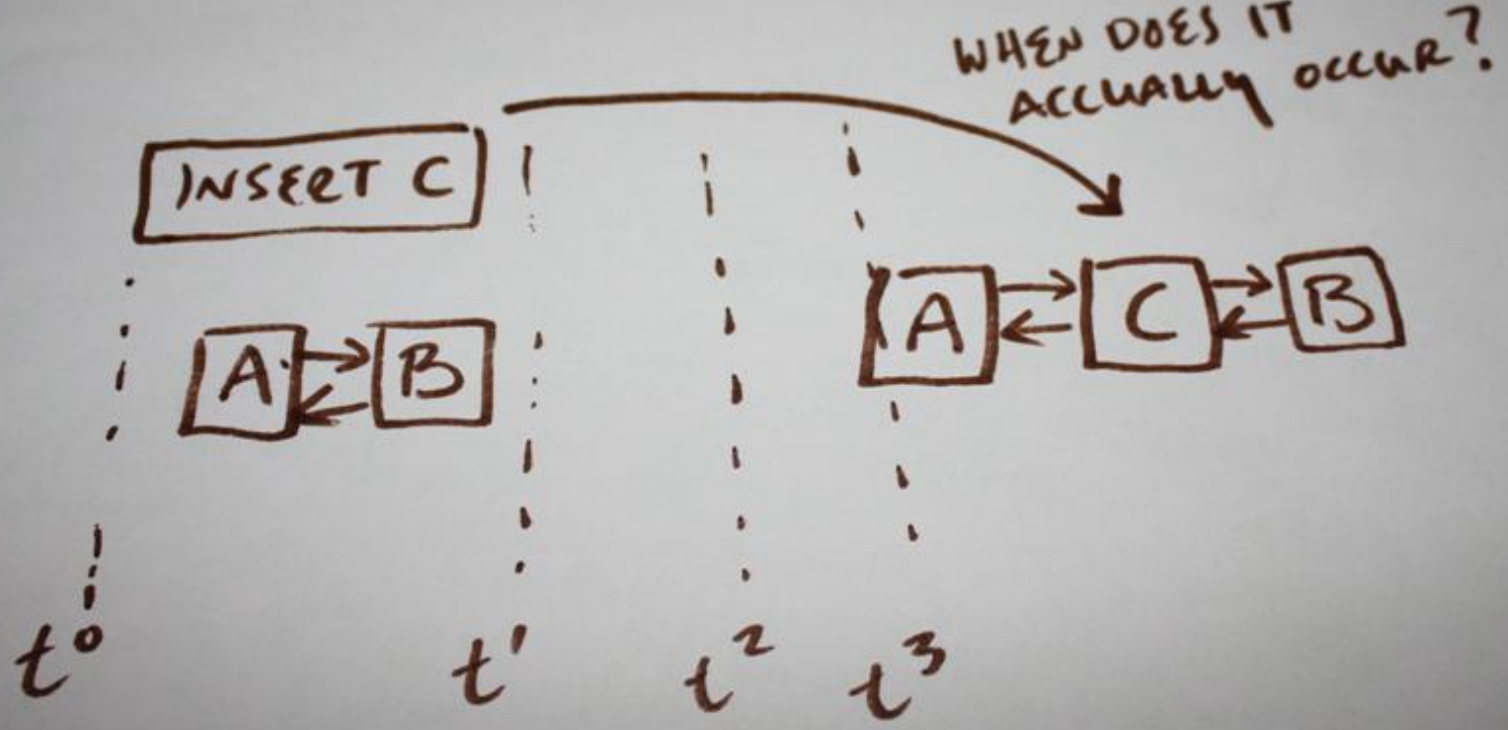
Which is necessary to define in concurrent problems.



define in
concurrent
problems.

How would
this make
the insert
OP Simpler?

WHAT ARE THE LATENCY REQUIREMENTS OF THE INSERT OP?



LESSON

DOUBLY-LINKED LIST IS SEQUENTIAL DATA STRUCT.

PRESUMES:

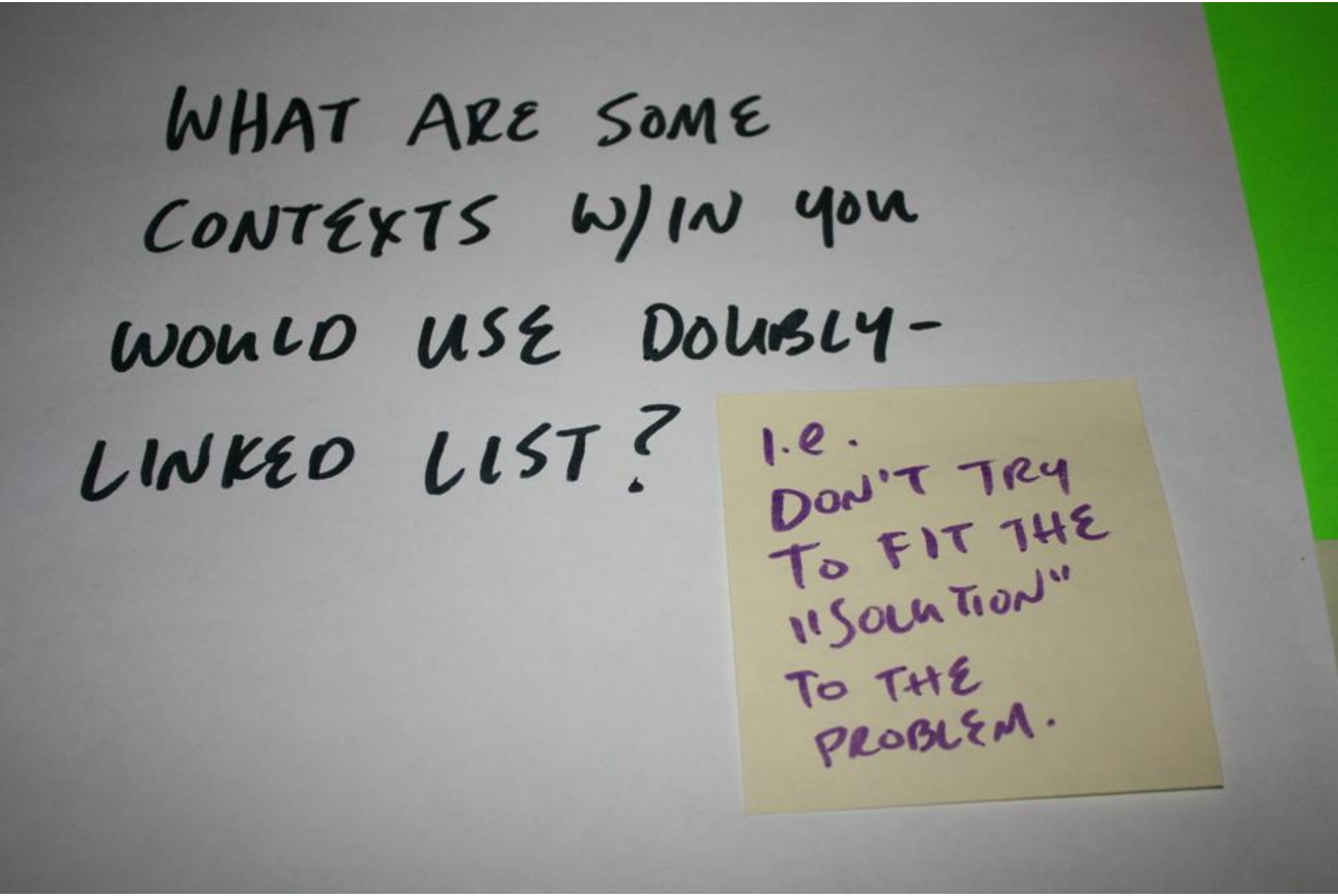
- ZERO LATENCY
- GUARANTEED ORDER (SEQUENTIAL)

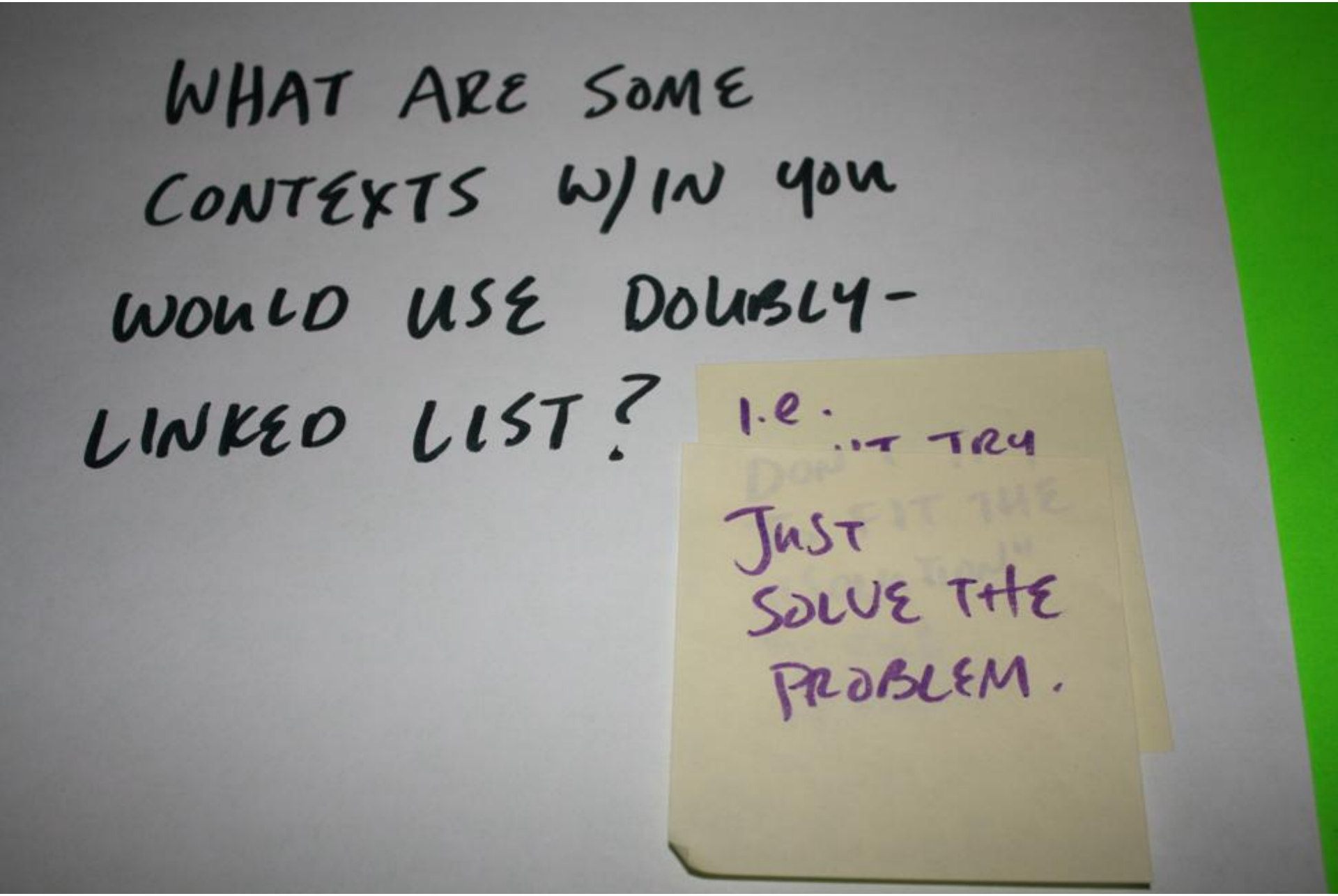
CONCURRENT DATA STRUCTURES DEFINED BY:

- EXPLICIT LATENCY REQUIREMENTS

- EXPLICIT ORDERING RULES

WHAT ARE SOME CONTEXTS W/IN YOU WOULD USE DOUBLY-LINKED LIST?


WHAT ARE SOME CONTEXTS W/IN YOU WOULD USE DOUBLY-LINKED LIST?
I.e. DON'T TRY TO FIT THE "SOLUTION" TO THE PROBLEM.

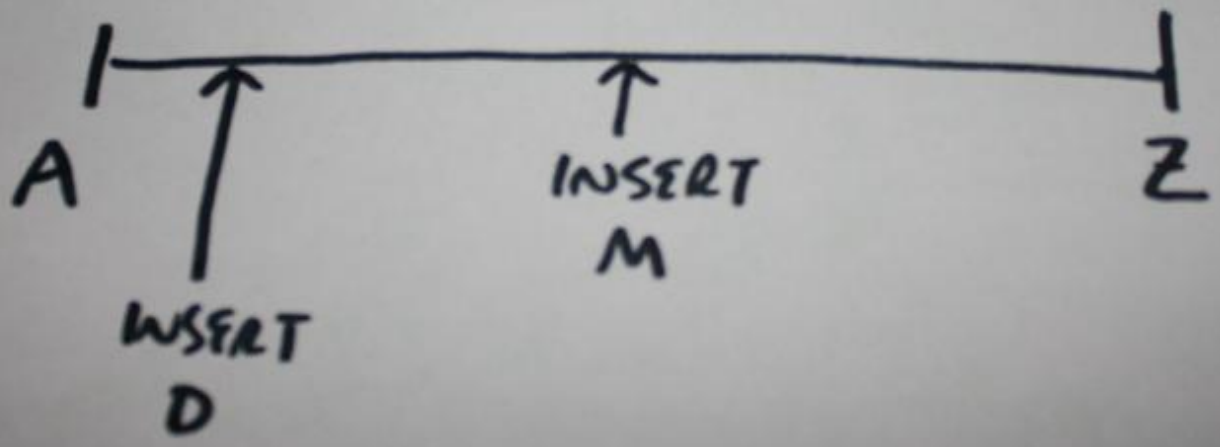

WHAT ARE SOME CONTEXTS W/IN YOU WOULD USE DOUBLY-LINKED LIST?
I.e. ... TRY JUST SOLVE THE PROBLEM.

e.g. Insert Sort

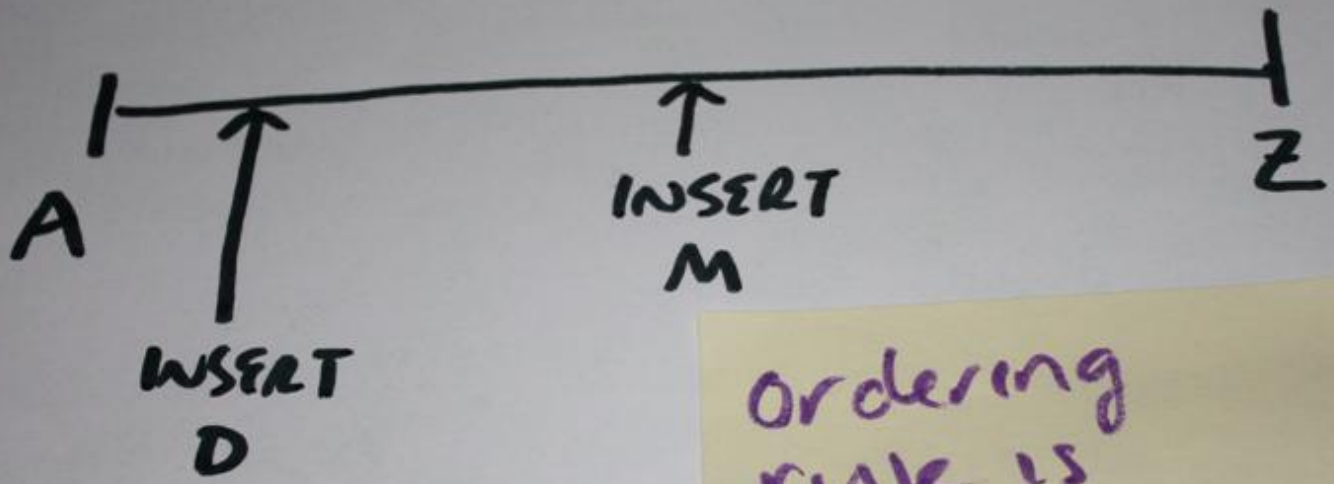
INSERT SORT MEANS

INSERT AT POSITION IN

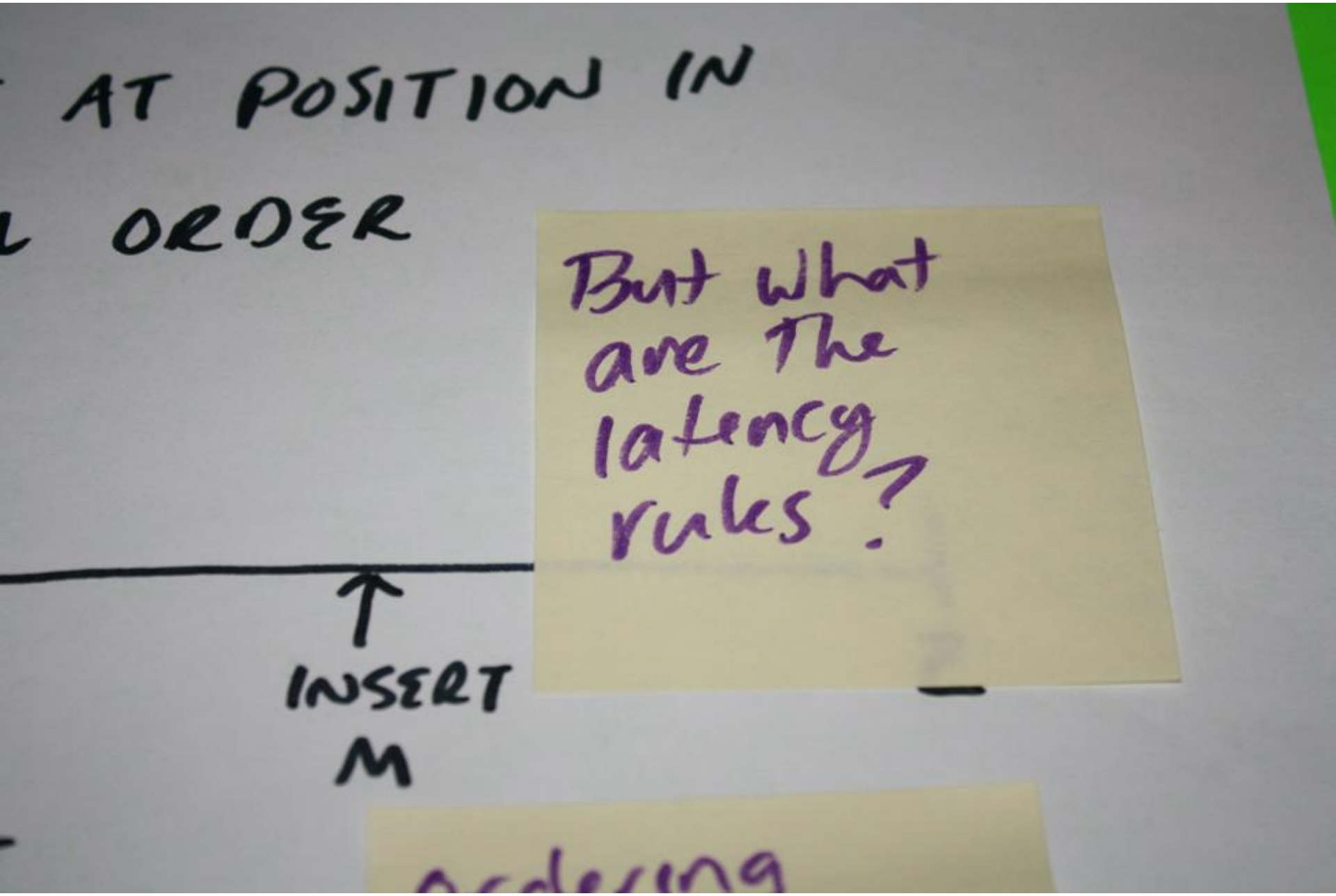
GLOBAL ORDER

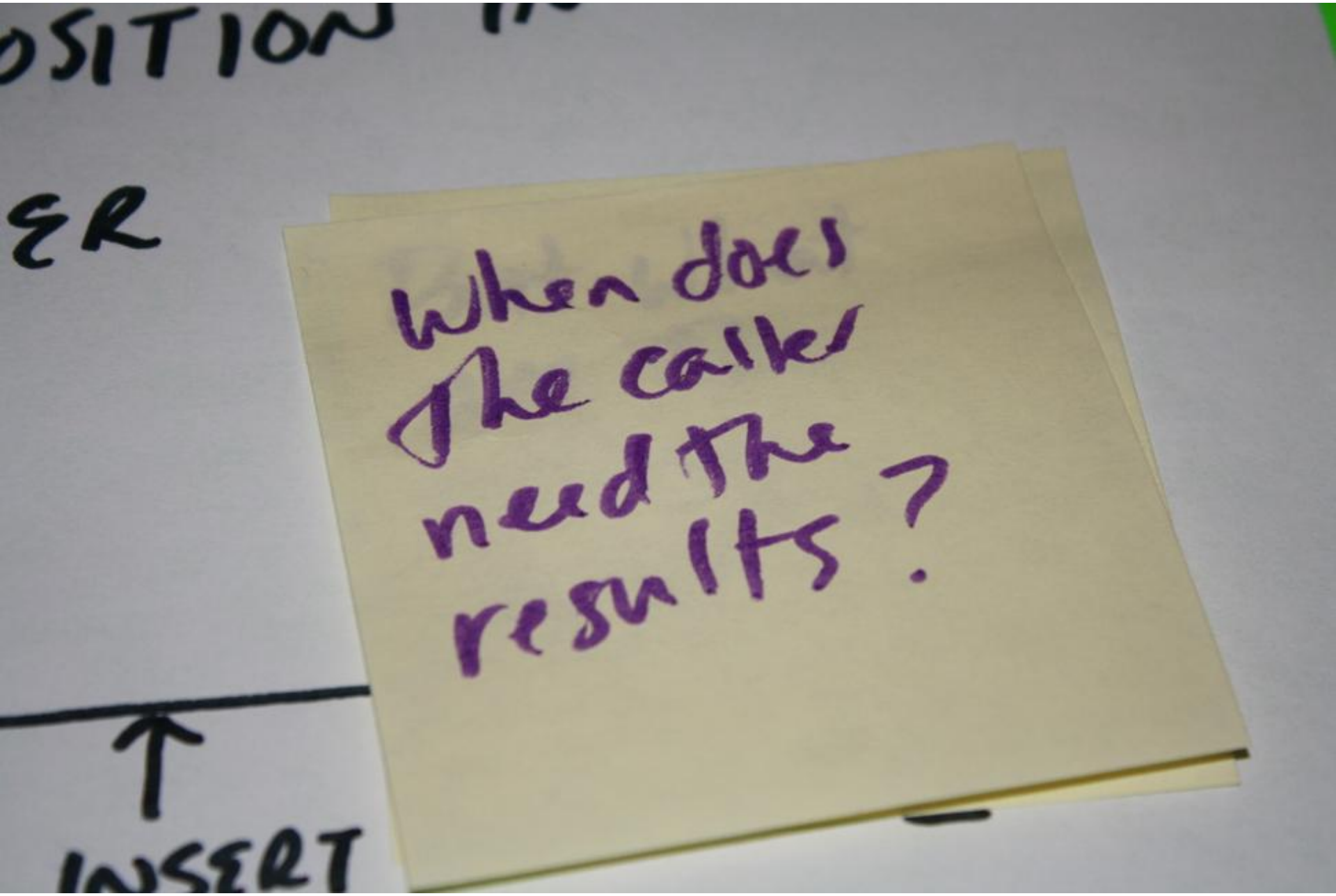


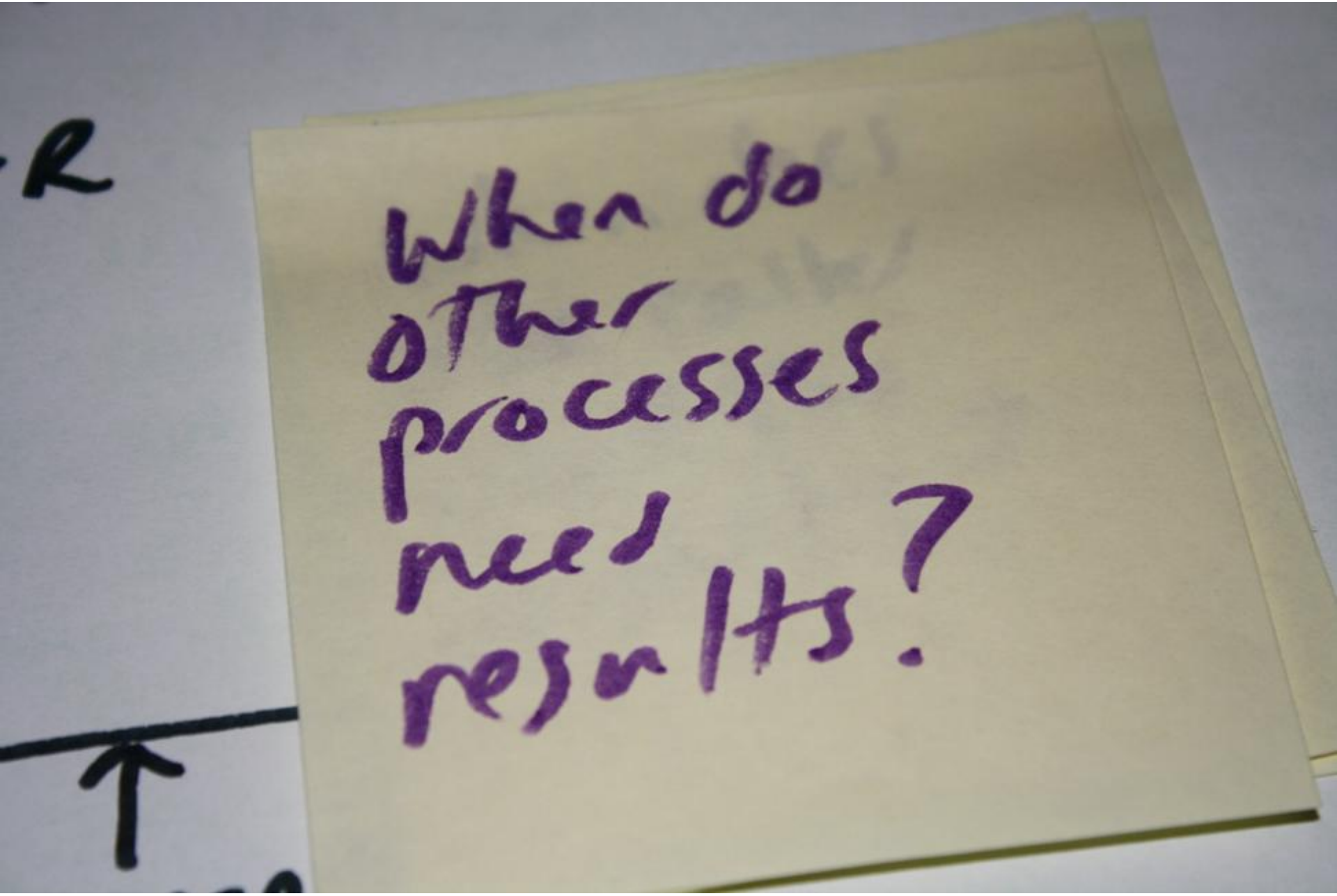
GLOBAL



ordering rule is global sorting / compare func.

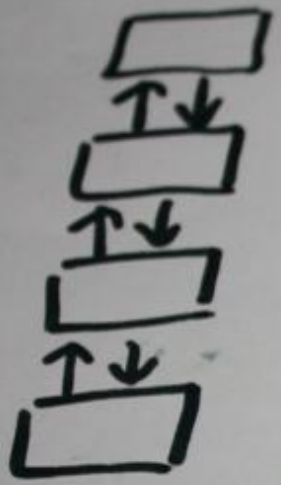

AT POSITION IN
ORDER
But what are the latency rules?
INSERT
M


When does the caller need the results?

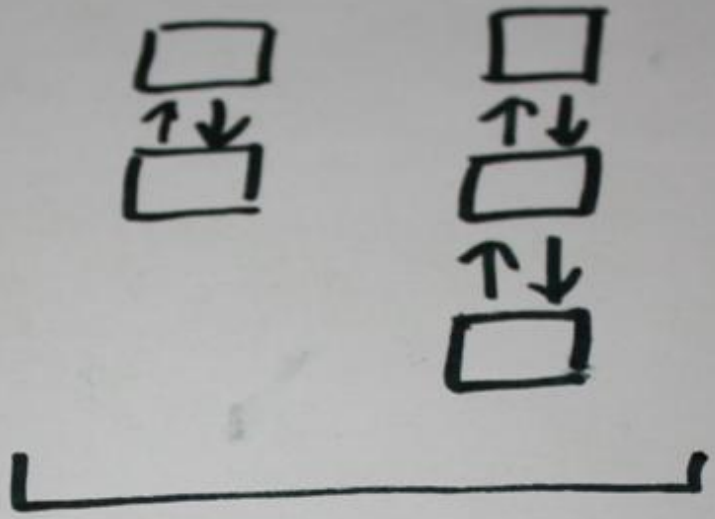

When do other processes need results?


GLOBAL ORDE
Do you think dbl linked list will be right data?
INSERT

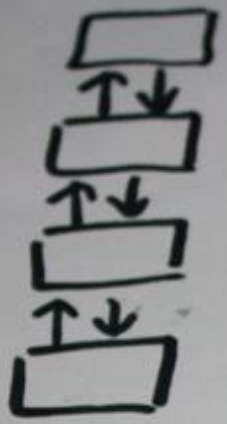
e.g. Resource Mgmt w/ variable life times



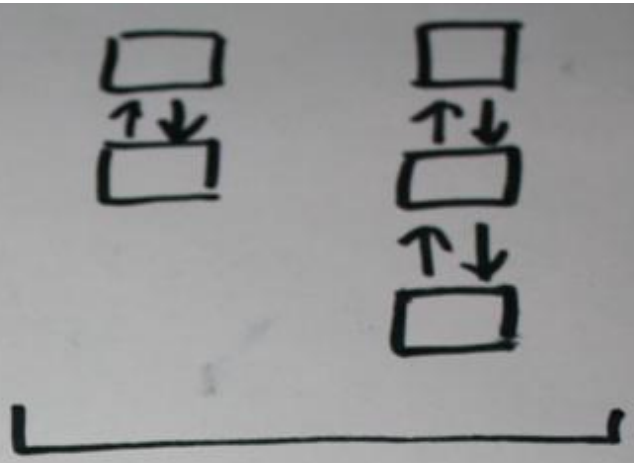
LIMITED RESOURCE LIST



- SPARSE LISTS OF ALLOCATED NODES
- VARIABLE LIFETIMES
- DIFF "OWNERS", DIFF LISTS

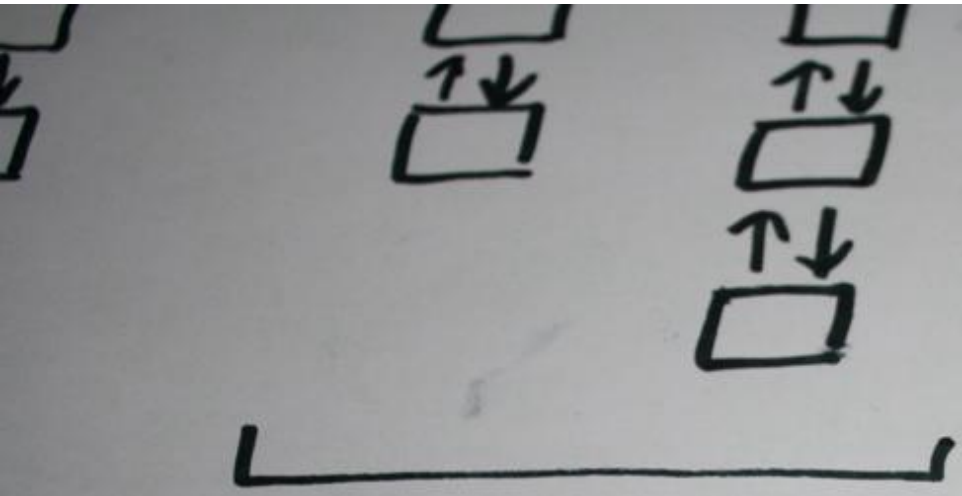


LIMITED RESOURCE LIST



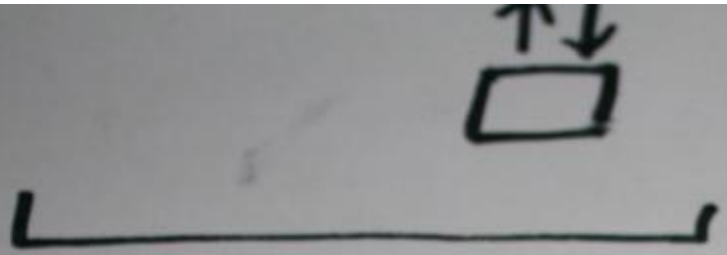
- SPARSE LISTS OF ALLOCATED NODES
- VARIABLE LIFETIMES
- DIFF "OWNERS", DIFF LISTS

↑ START W/ LIST OF RESOURCES



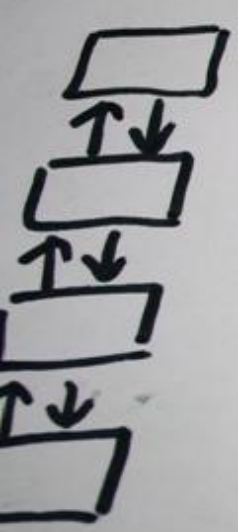
- SPARSE LISTS OF ALLOCATED NODES
- VARIABLE LIFETIMES
- DIFF "OWNERS", DIFF LISTS

← As they're alloc'd, add to new "owner" lists.



- SPARSE LISTS OF ALLOCATED NODES
- VARIABLE LIFETIMES
- DIFF "OWNERS", DIFF LISTS

← When "deleted" remove from list, return to res. list.

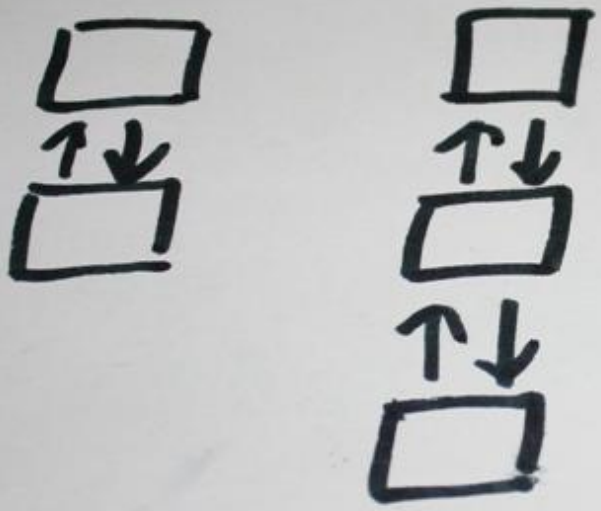


← APPEND TO END. ORDER NOT IMPORTANT

TEO
URCE

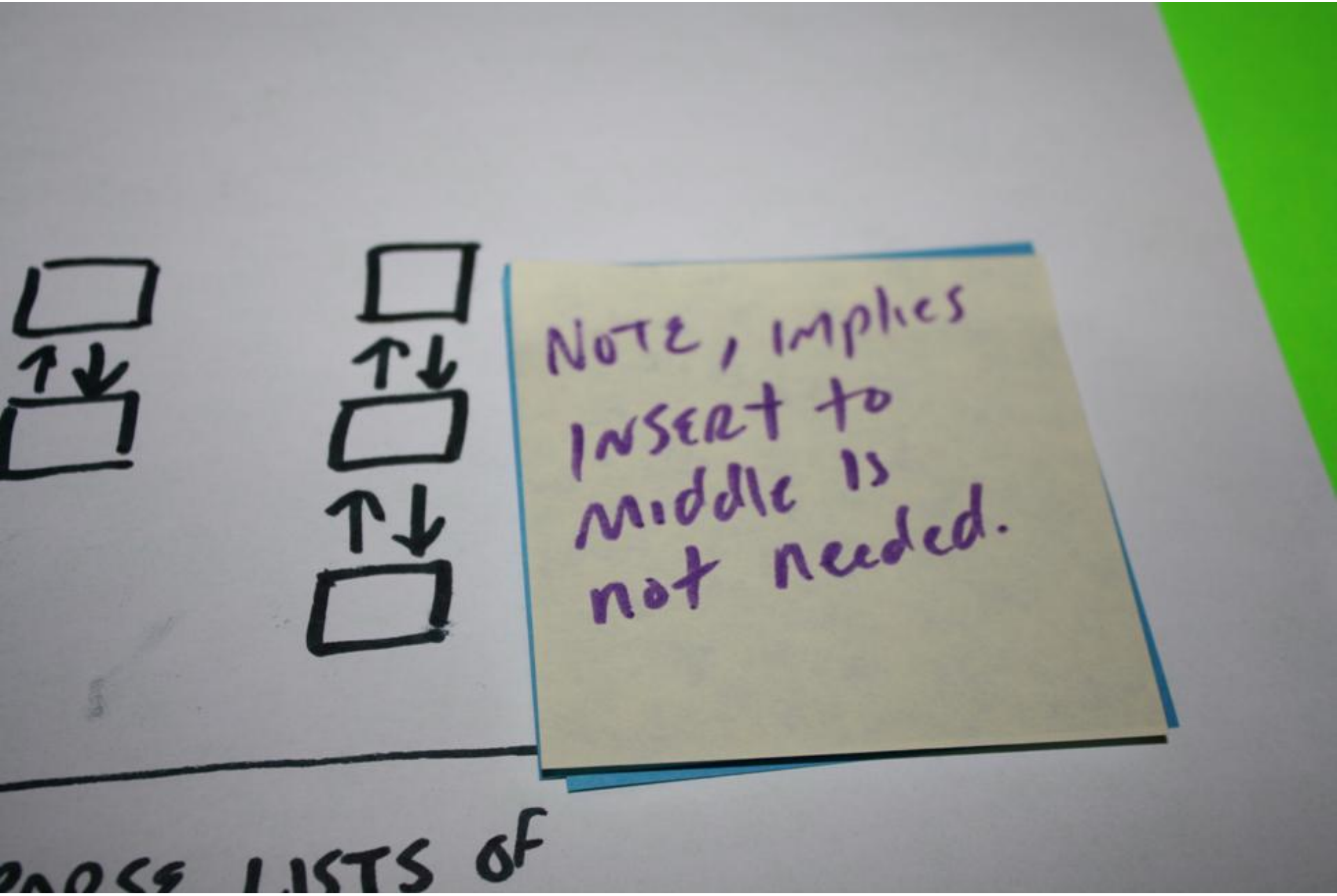
- SPARSE LISTS OF ALLOCATED NODES

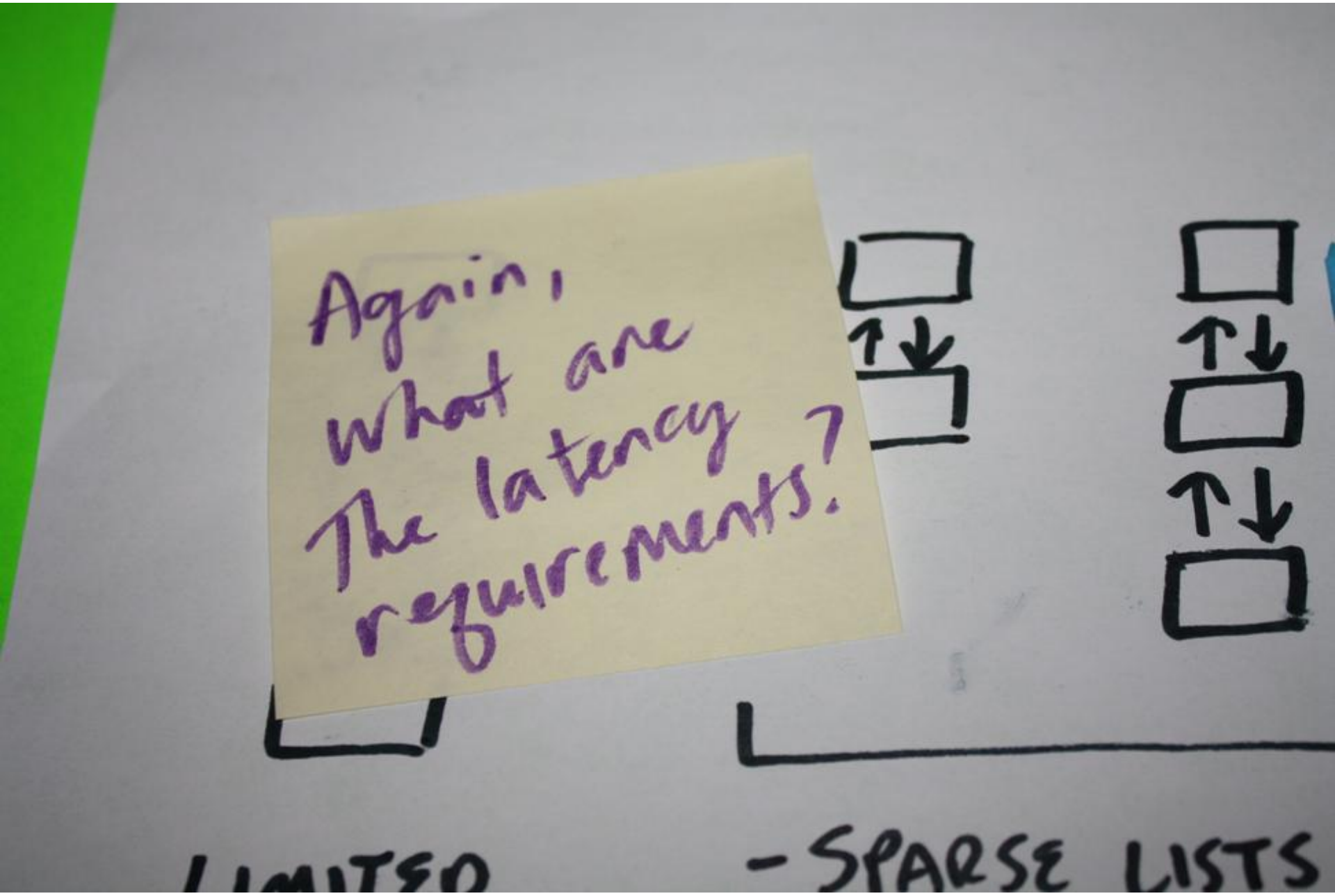
- VARIABLE LIFETIMES



THIS IS THE EXPLICIT ORDERING RULE

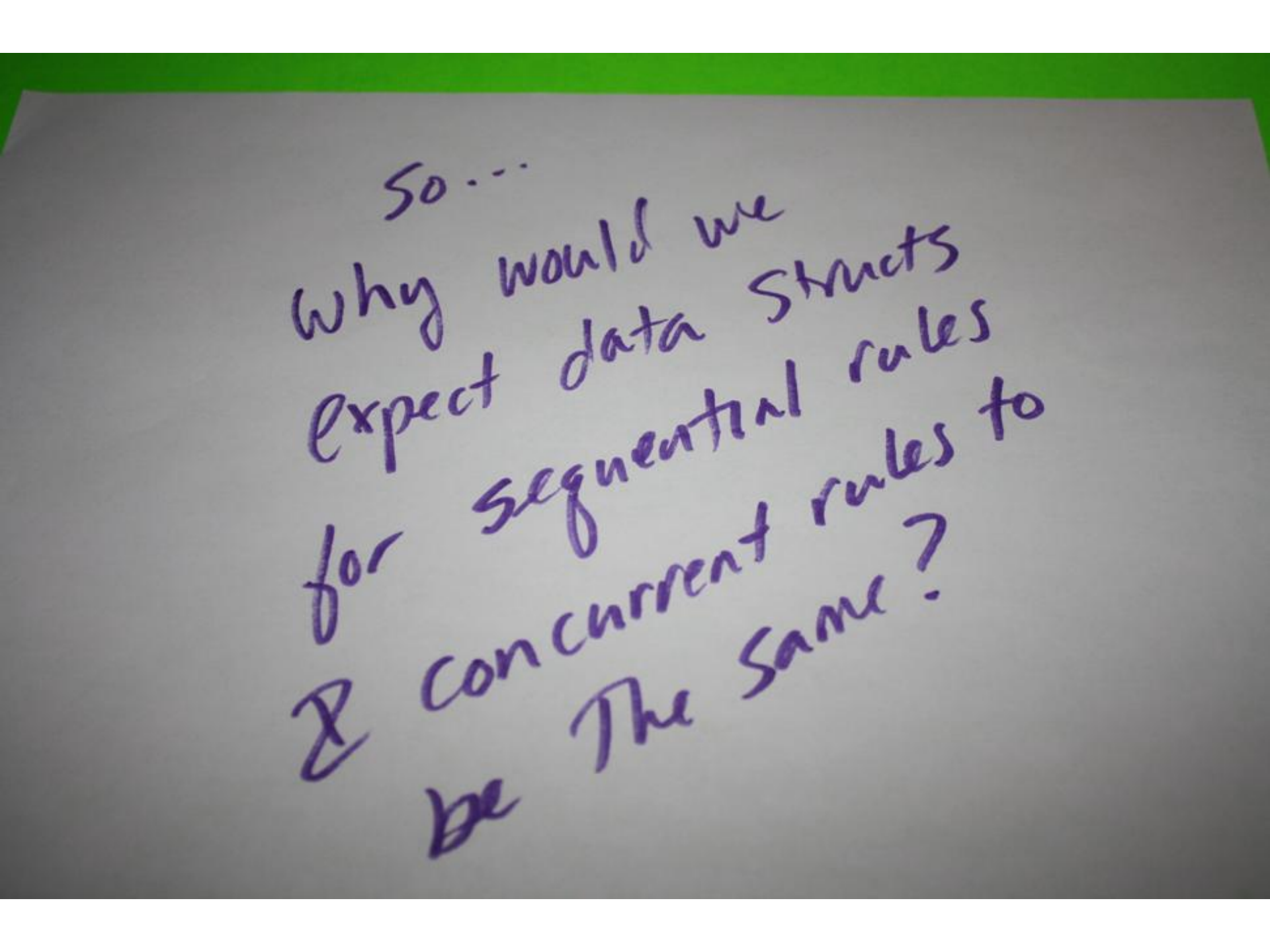
- SPARSE LISTS OF ALLOCATED NODES

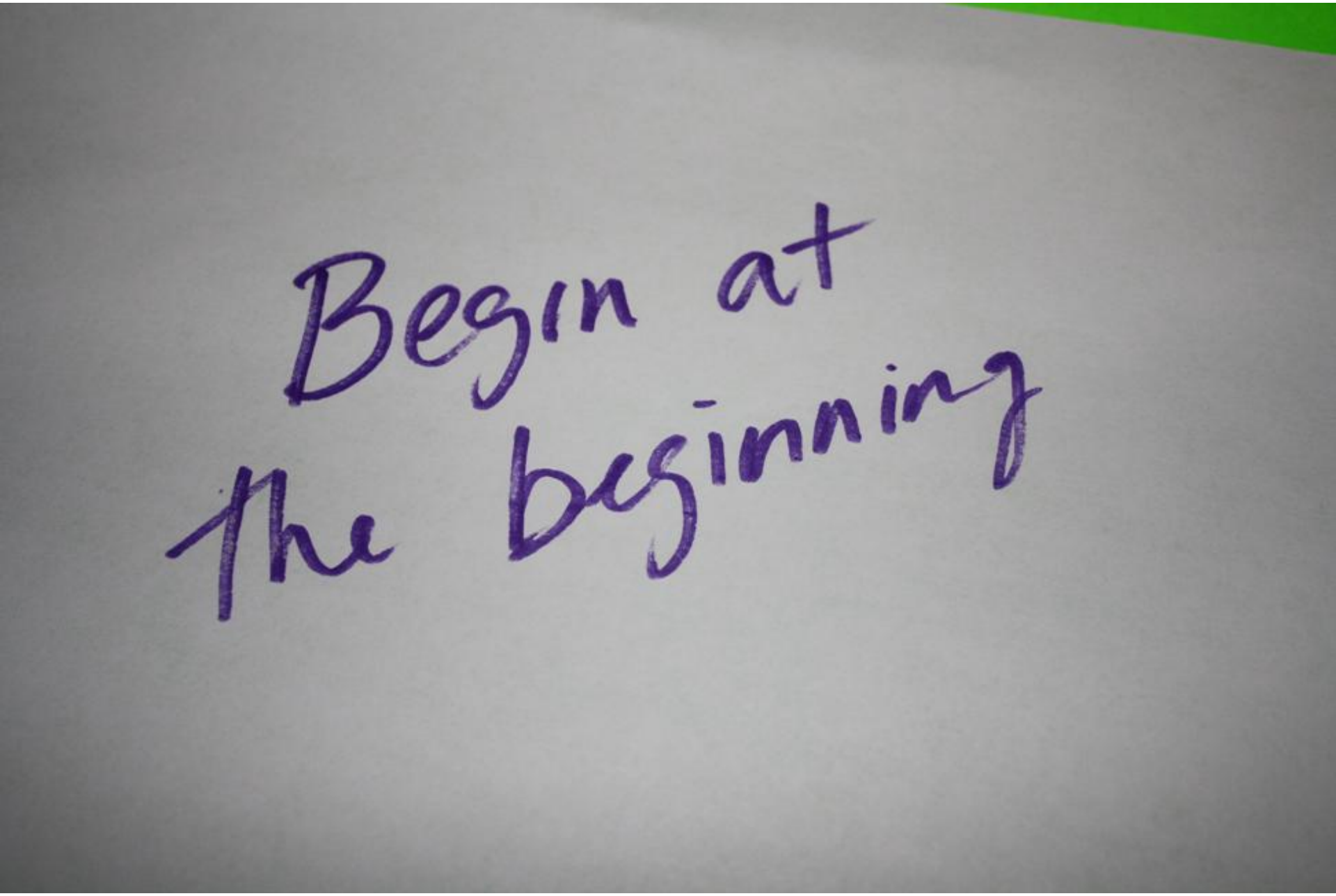

Note, implies
INSERT to
Middle is
not needed.
LISTS OF


Again,
What are
The latency
requirements?
- SPARSE LISTS

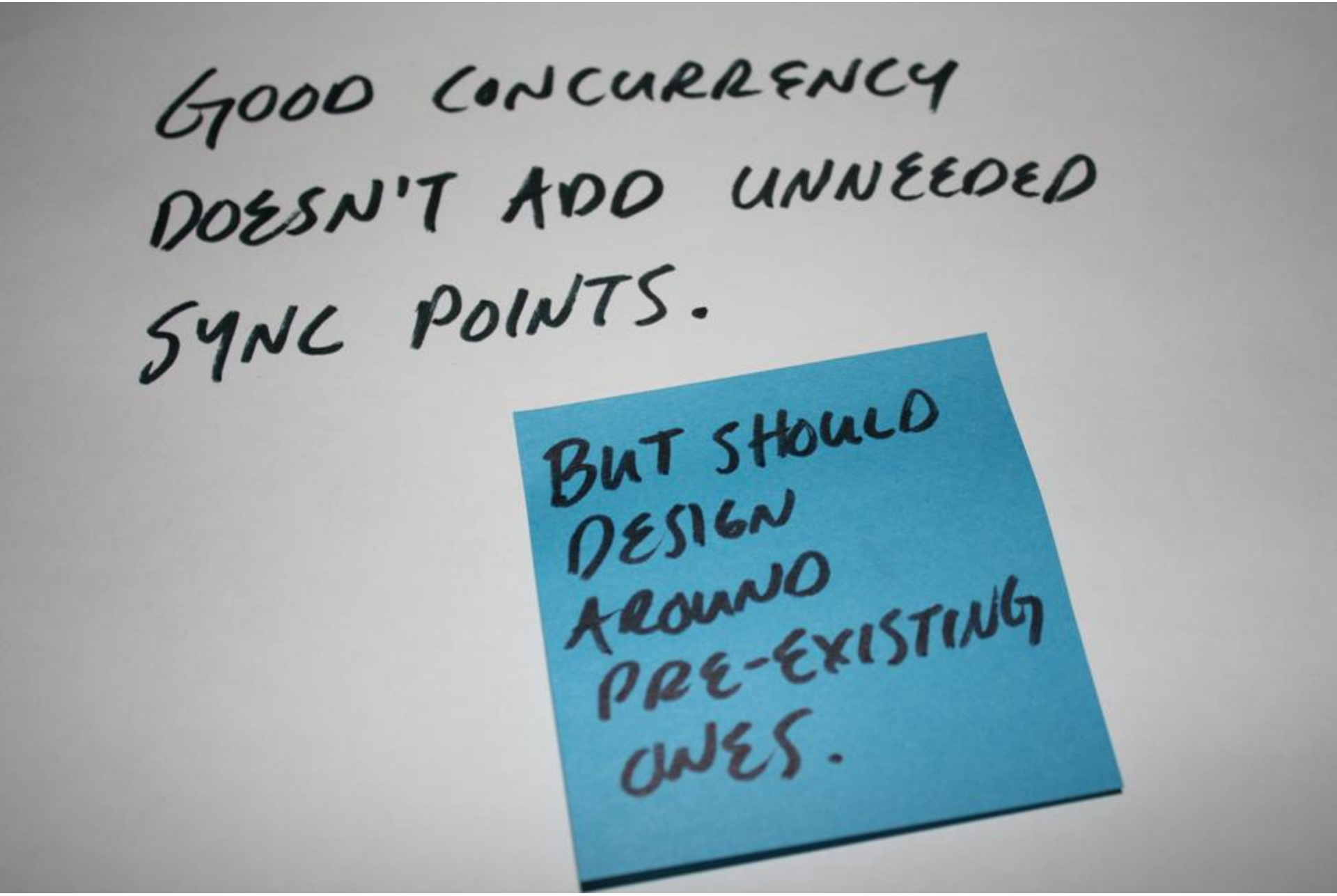
DATA FOR DIFFERENT ORDERING RULES WILL BE DIFFERENT.

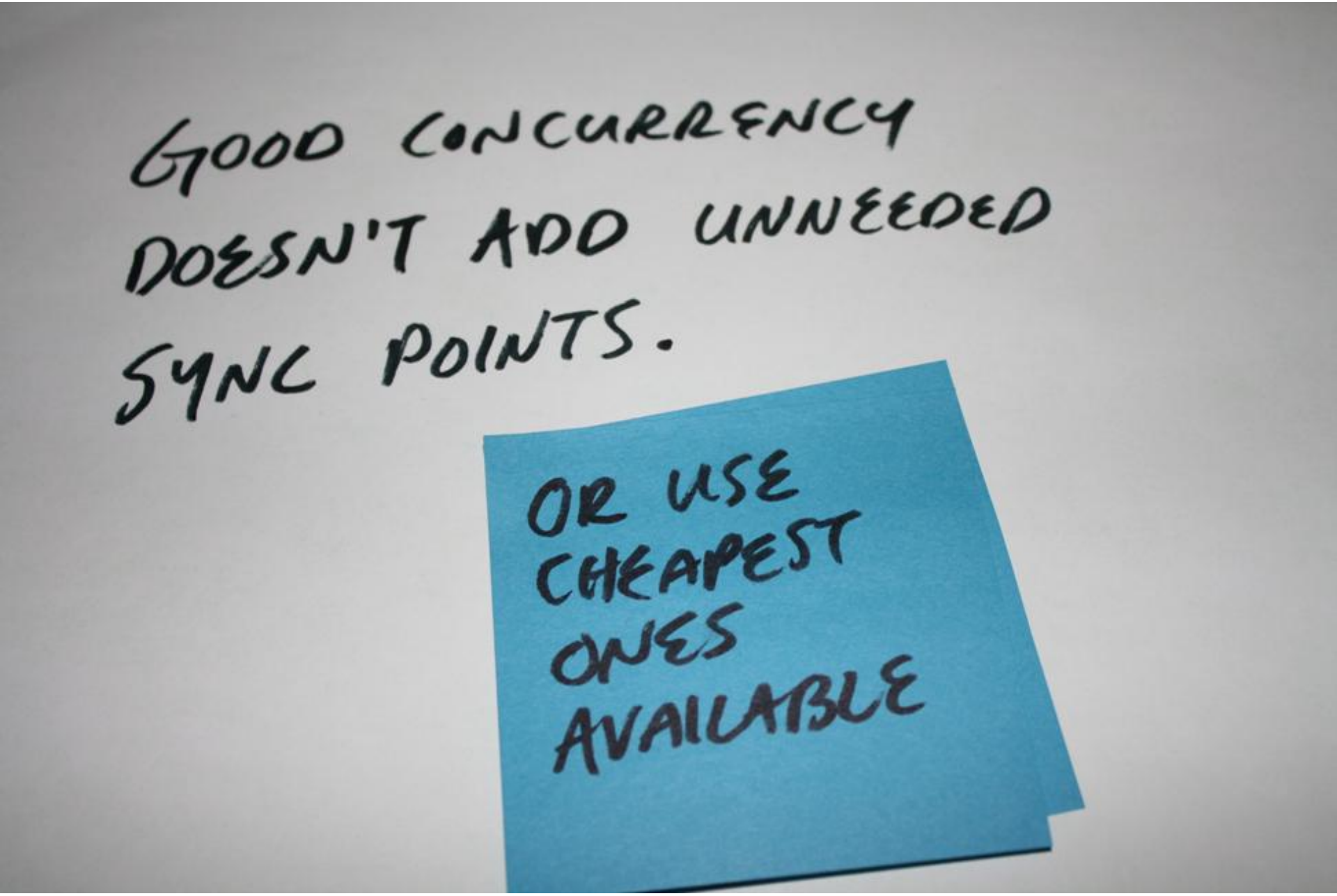
Data for different latency rules will be different.


So...
Why would we
expect data structs
for sequential rules
& concurrent rules to
be the same?


Begin at
the beginning

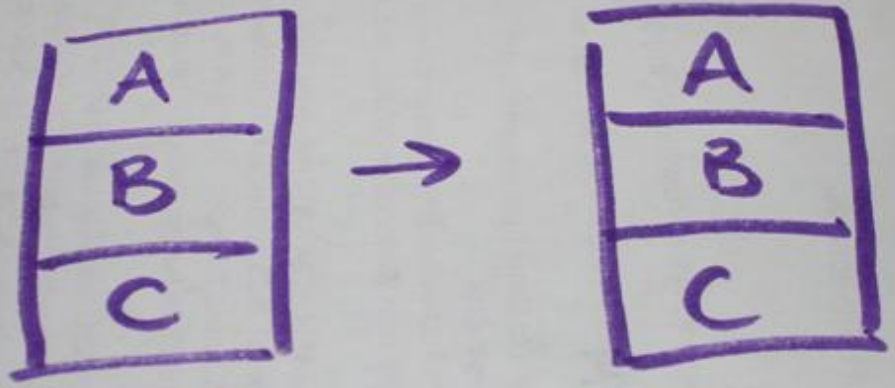
GOOD CONCURRENCY
DOESN'T ADD UNNEEDED
SYNC POINTS.


GOOD CONCURRENCY DOESN'T ADD UNNEEDED SYNC POINTS.
BUT SHOULD DESIGN AROUND PRE-EXISTING ONES.


GOOD CONCURRENCY
DOESN'T ADD UNNEEDED
SYNC POINTS.
OR USE
CHEAPEST
ONES
AVAILABLE

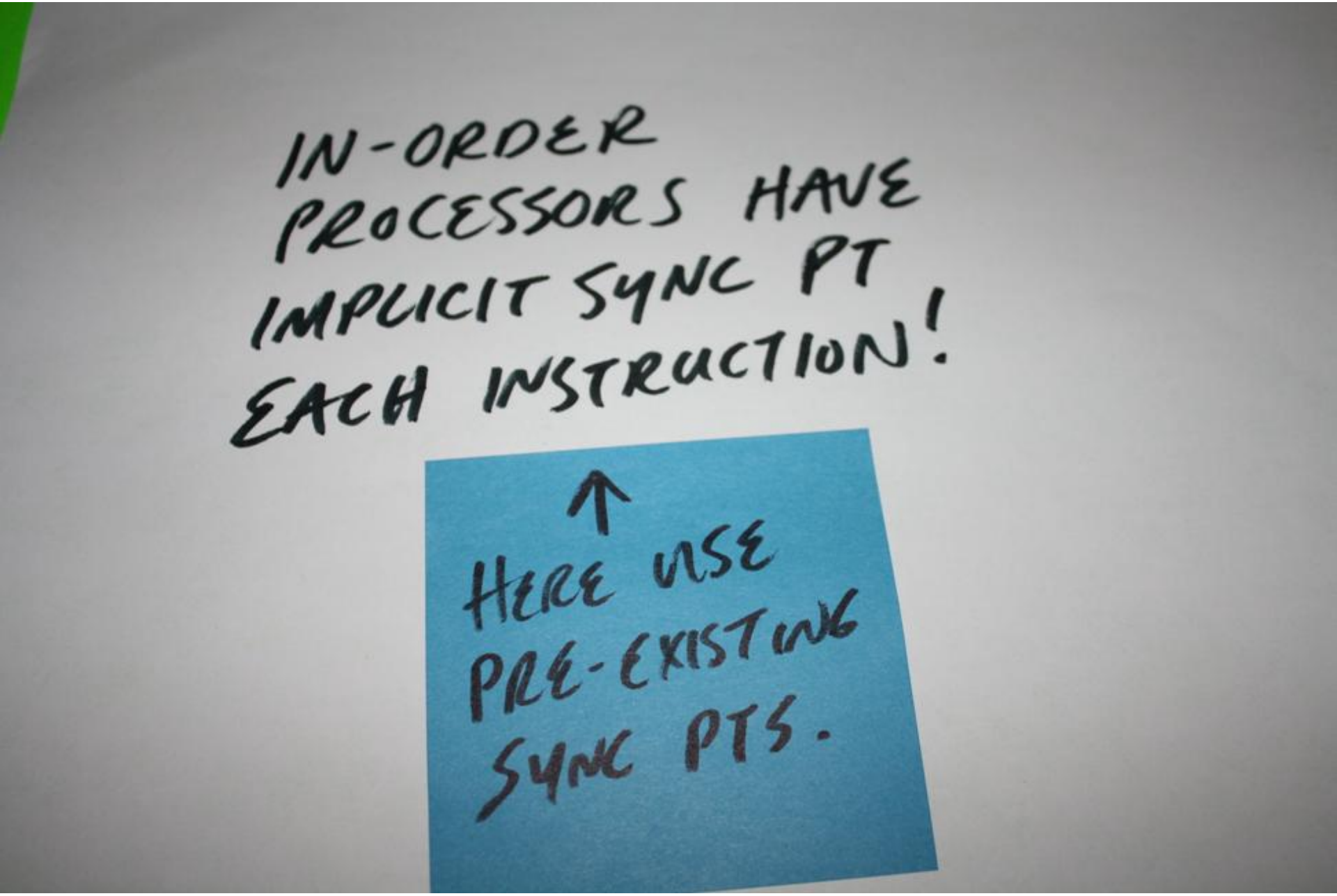
Concurrency starts with atomic transaction

IN-ORDER STORES

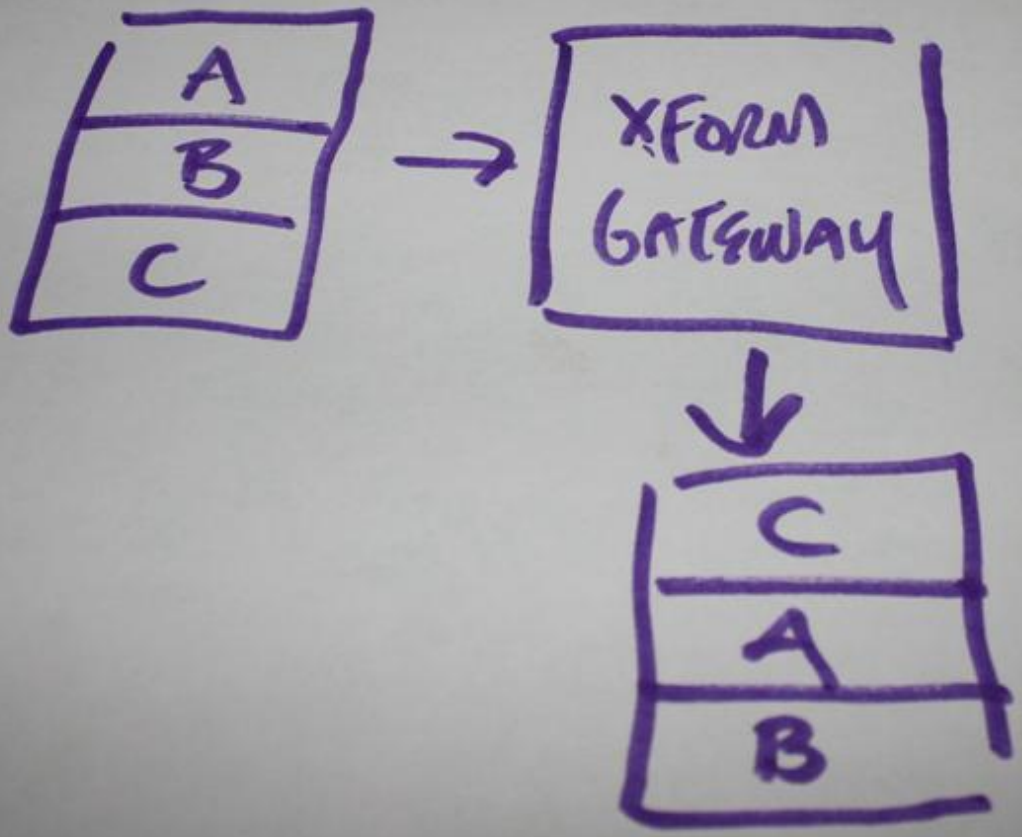


FIFO, NO CHANGE

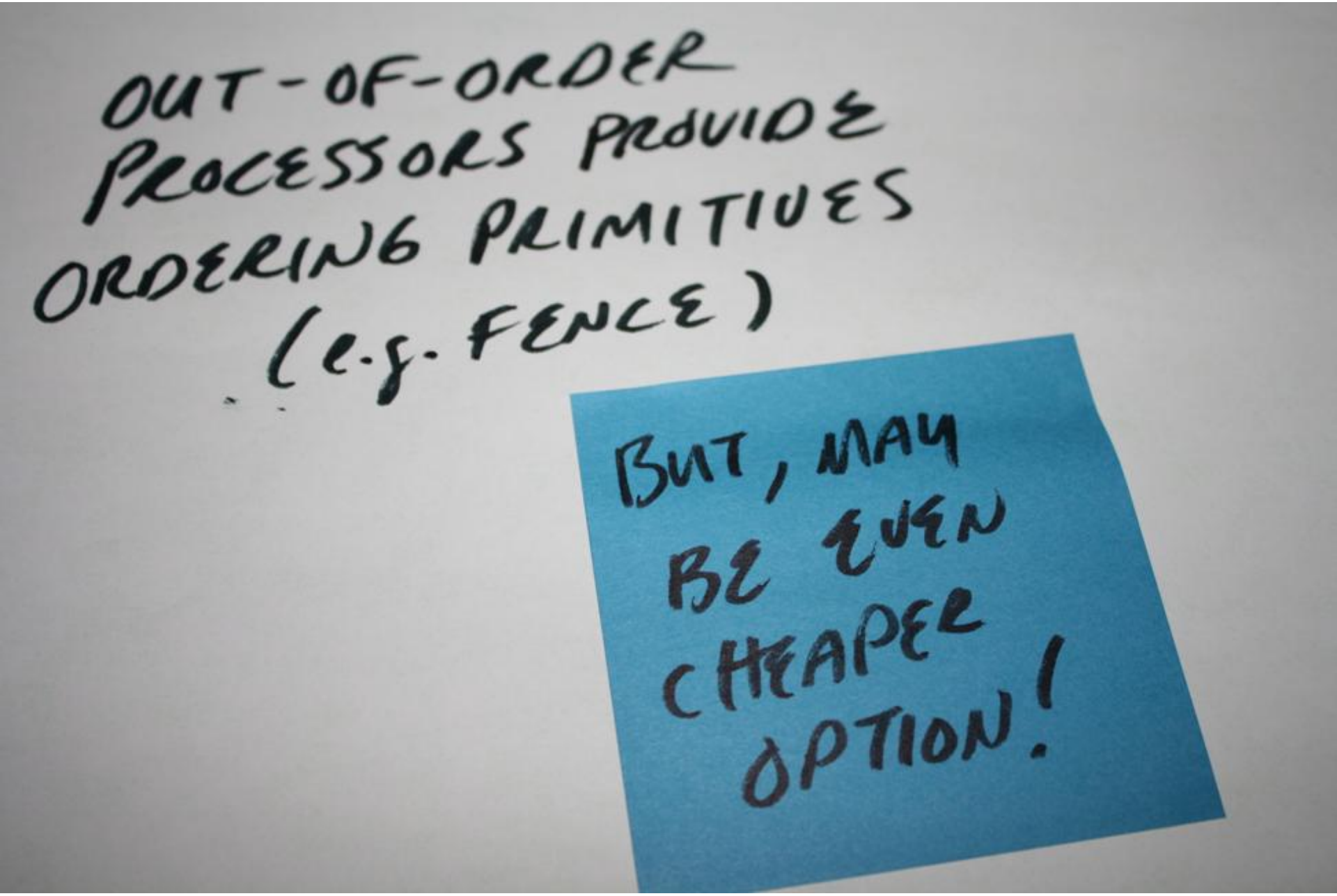
IN-ORDER PROCESSORS HAVE IMPLICIT SYNC PT EACH INSTRUCTION!

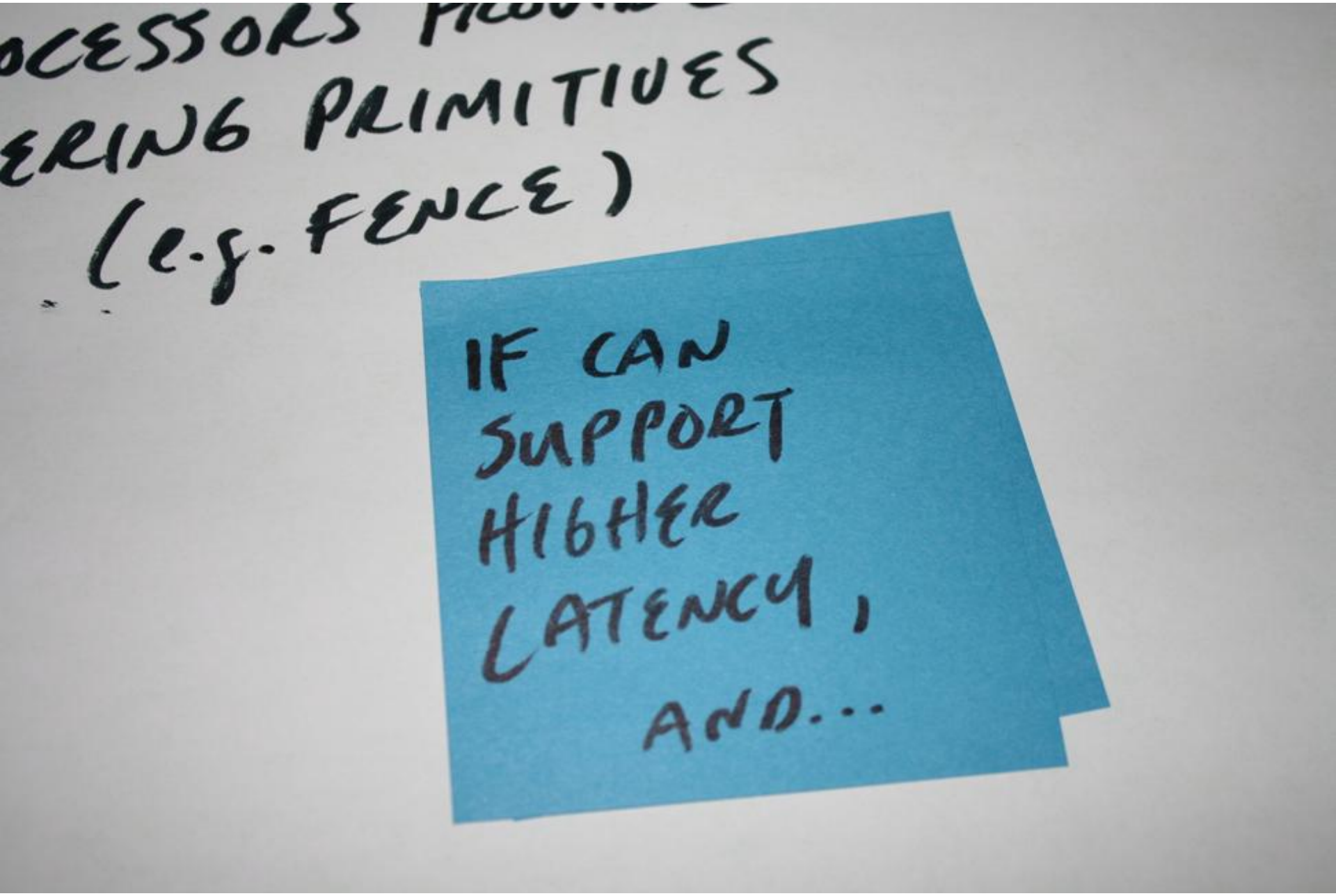

IN-ORDER
PROCESSORS HAVE
IMPLICIT SYNC PT
EACH INSTRUCTION!
↑
HERE USE
PRE-EXISTING
SYNC PTS.

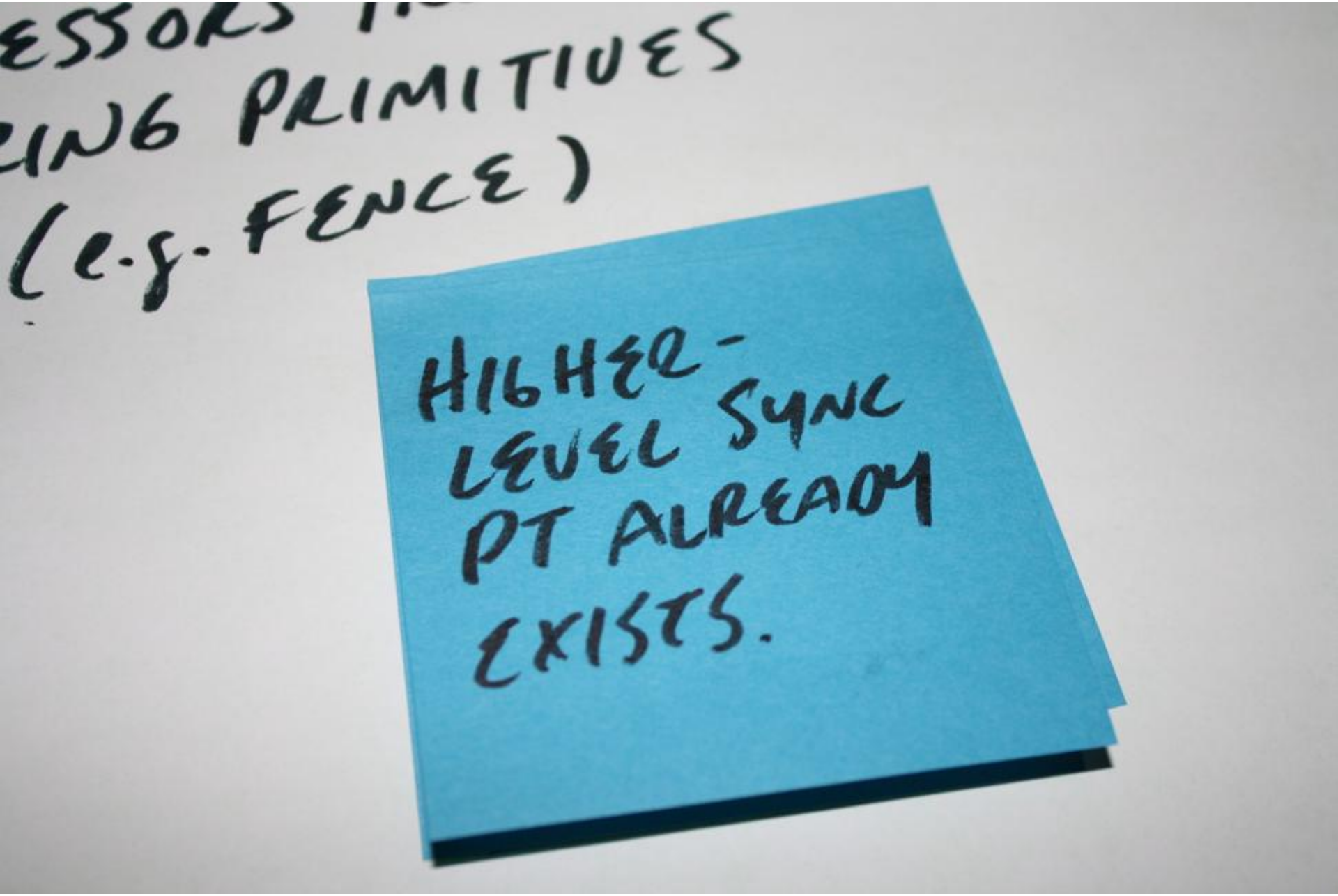
OUT-OF-ORDER STORES



OUT-OF-ORDER
PROCESSORS PROVIDE
ORDERING PRIMITIVES
(e.g. FENCE)


OUT-OF-ORDER
PROCESSORS PROVIDE
ORDERING PRIMITIVES
(e.g. FENCE)
BUT, MAY
BE EVEN
CHEAPER
OPTION!


OCESSORS
ERING PRIMITIVES
(e.g. FENCE)
IF CAN
SUPPORT
HIGHER
LATENCY,
AND...


ESSORS
ING PRIMITIVES
(E.G. FENCE)
HIGHER-
LEVEL SYNC
PT ALREADY
EXISTS.

ALSO NOTE

LANGUAGE/COMPILER

Compiler/optimizer re-orders instructions by definition!

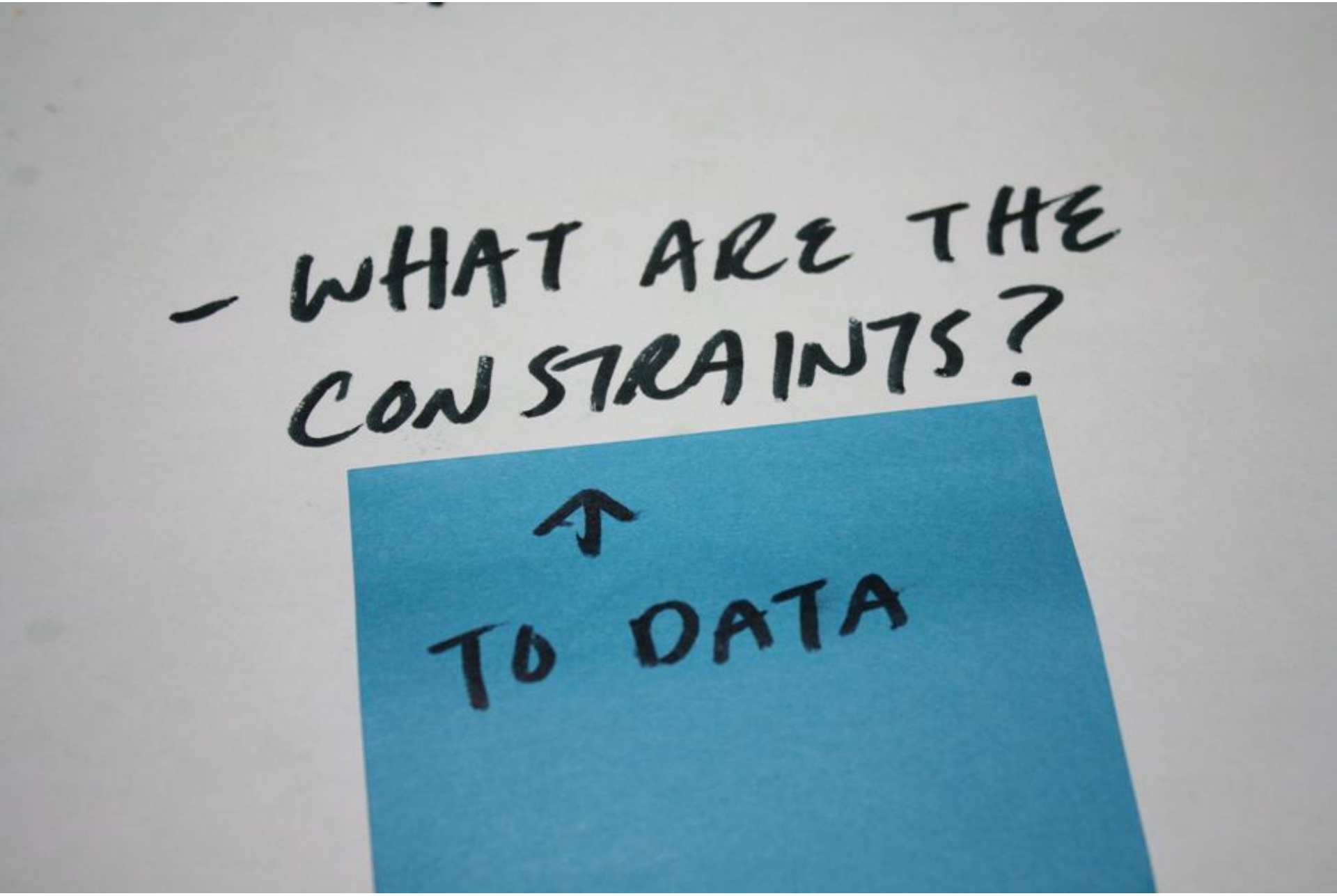
PROG. MUST force order!

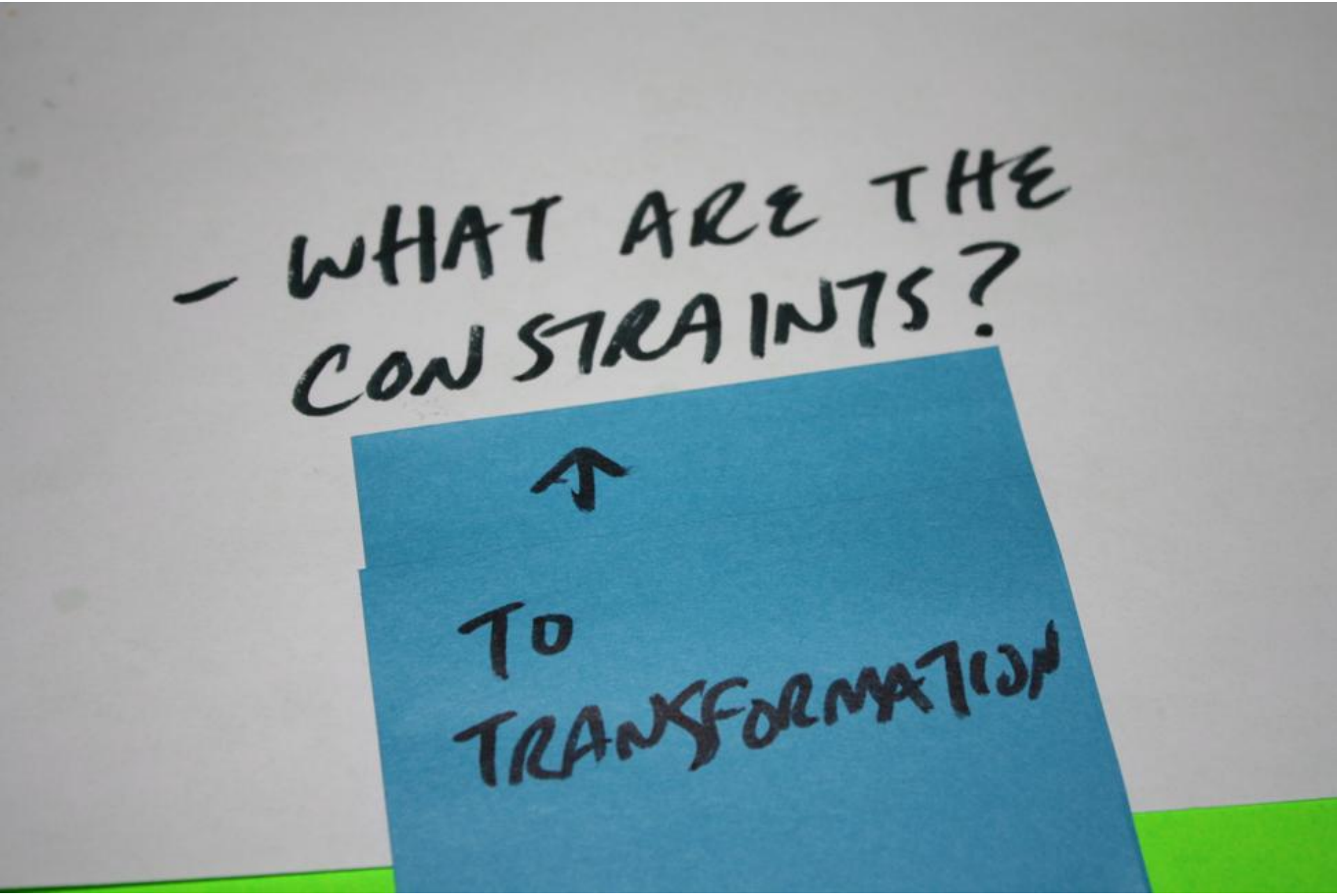
NEED TO ANSWER:

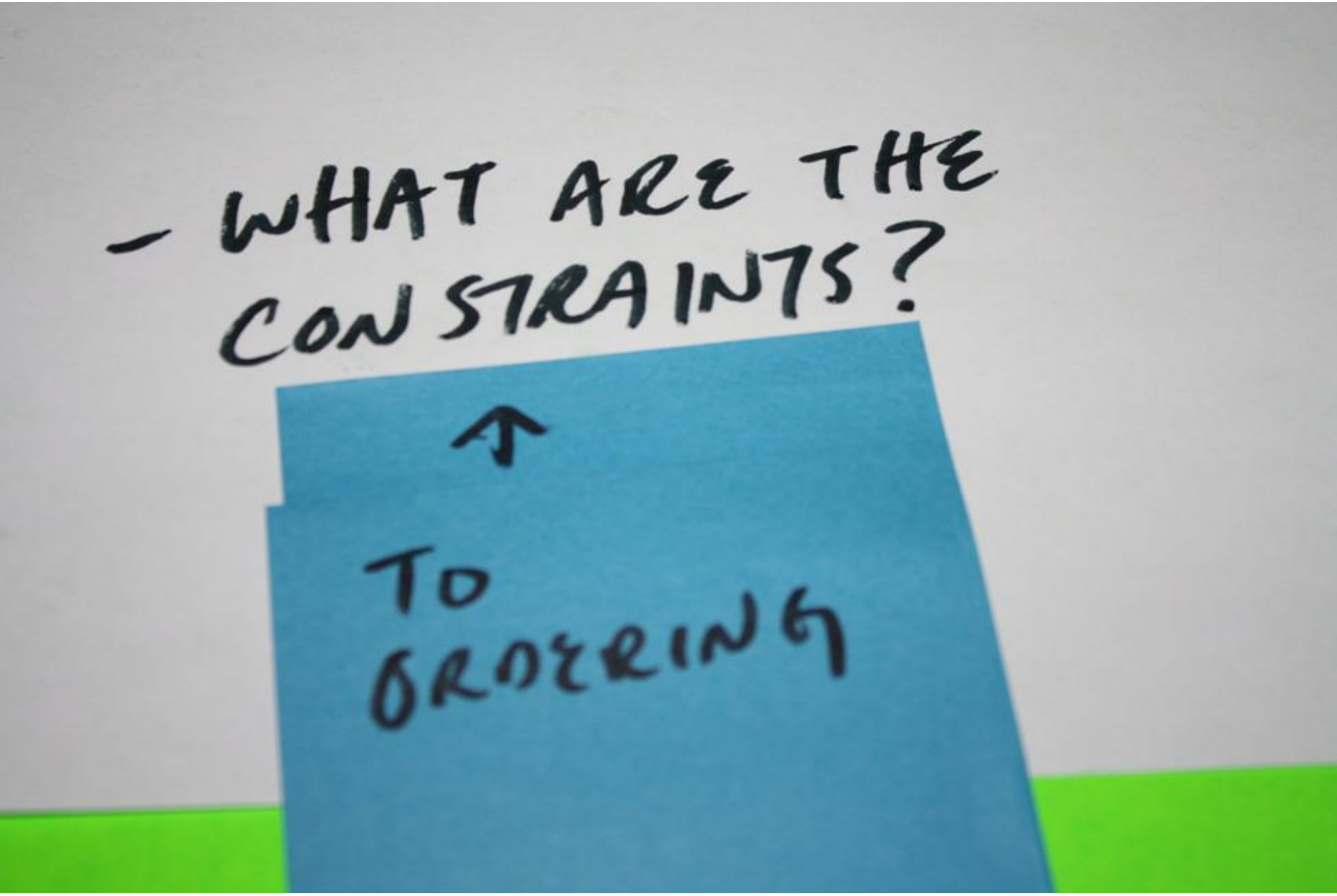
- HOW WILL DATA BE TRANSFORMED?

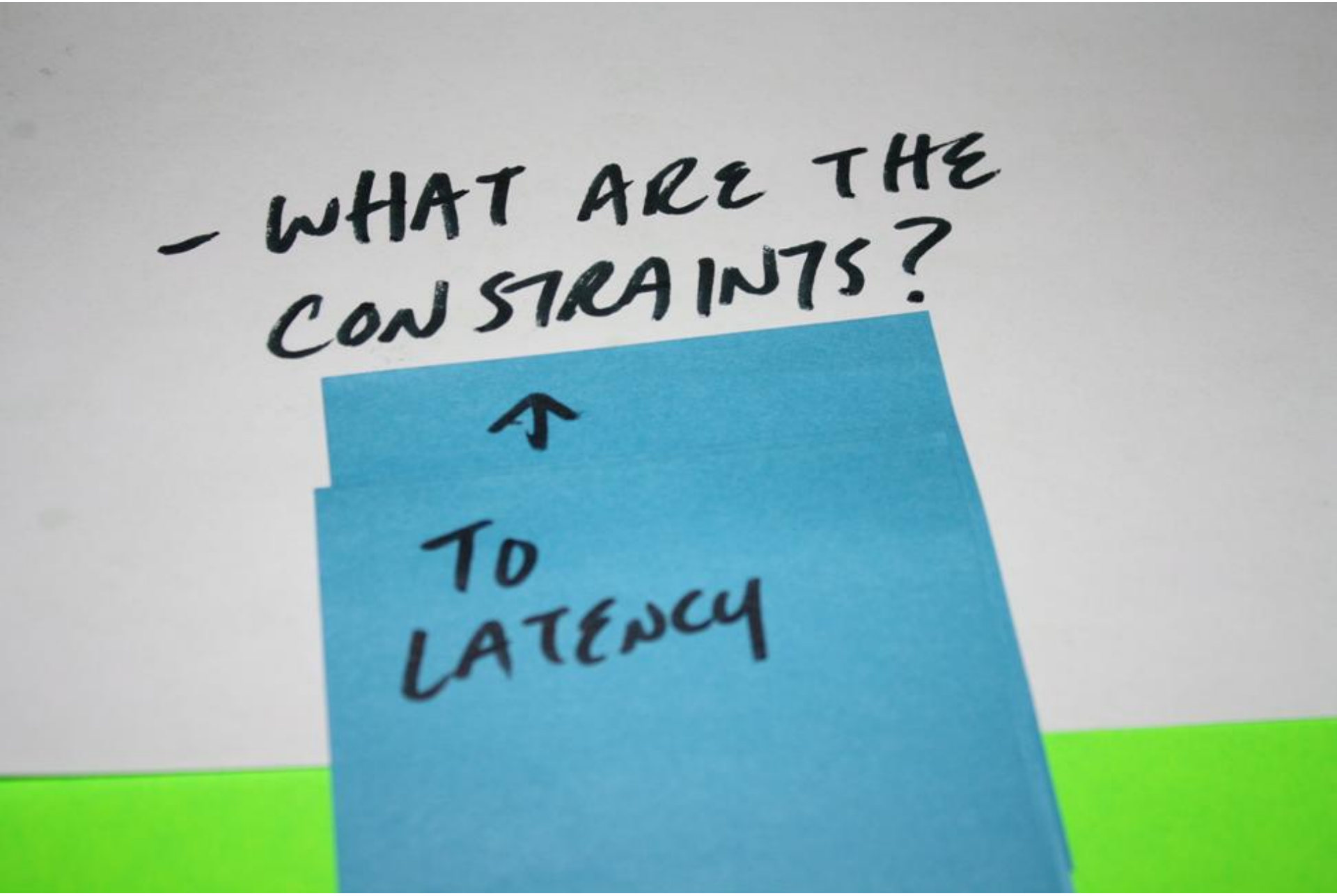
- WHAT ARE THE CONSTRAINTS?

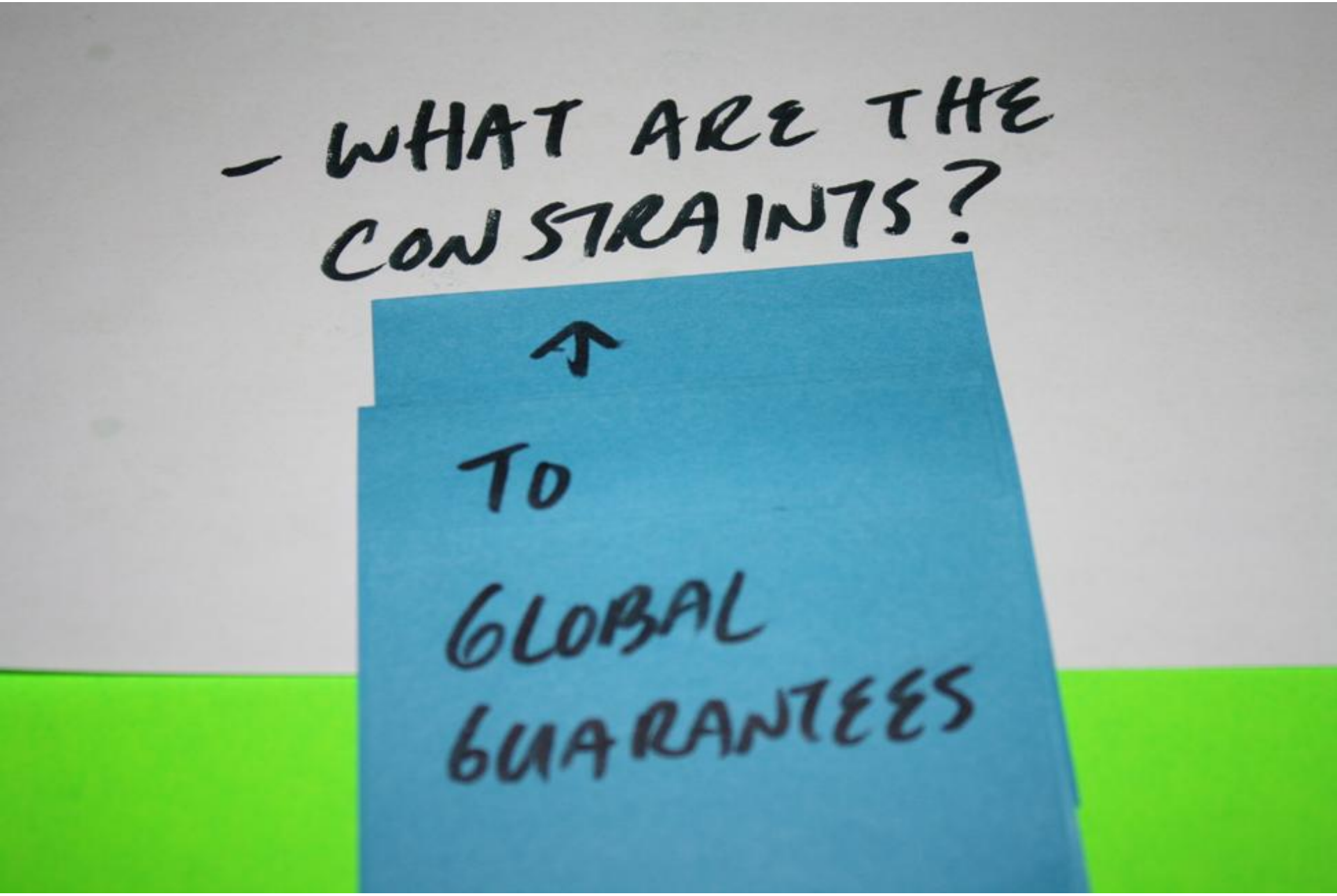

NEED TO ANSWER
- HOW WILL DATA BE TRANSFORMED?
↑
i.e.
OPERATIONS
- WH
Co

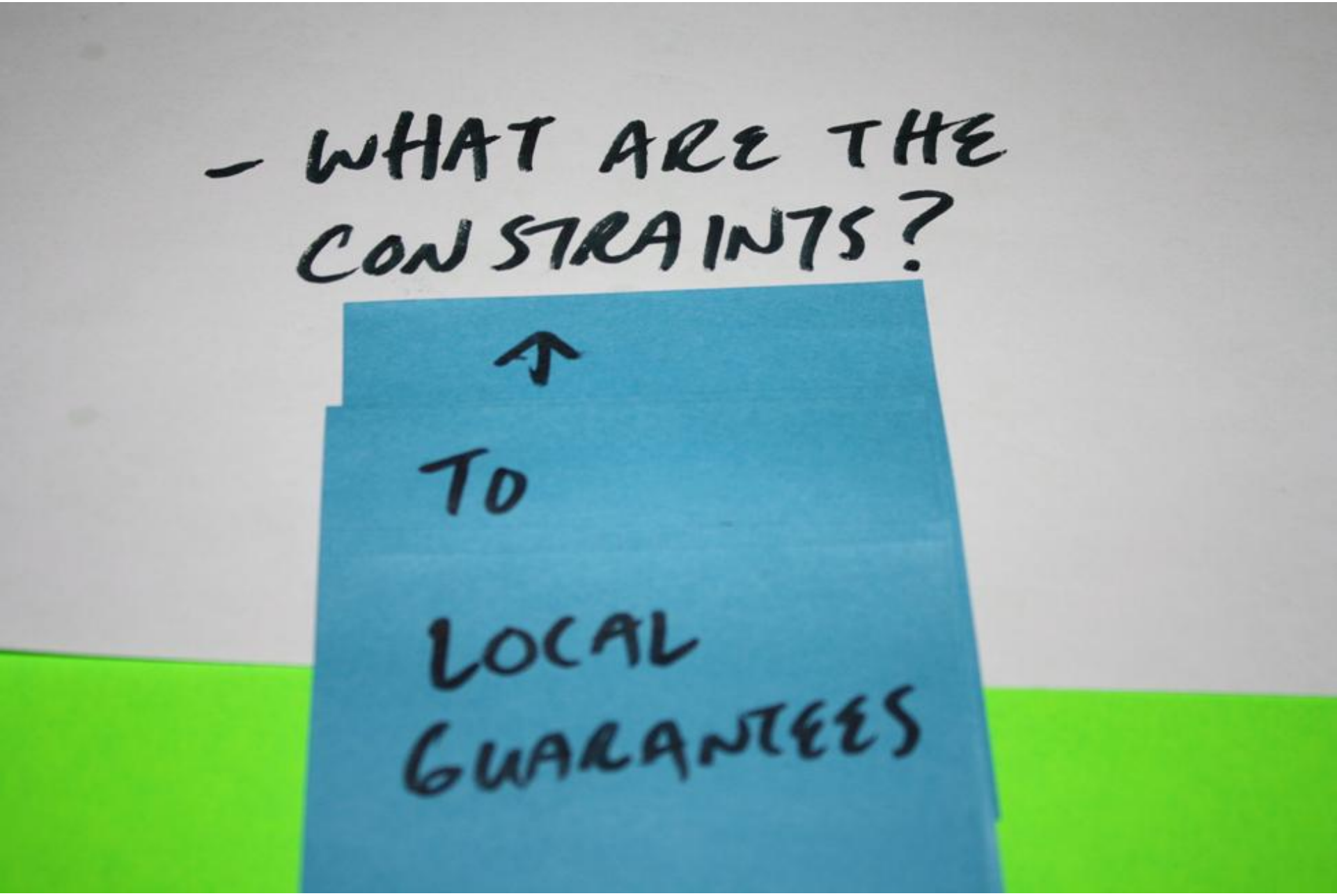

- WHAT ARE THE CONSTRAINTS?
↑
TO DATA


- WHAT ARE THE CONSTRAINTS?
↑
TO TRANSFORMATION


- WHAT ARE THE CONSTRAINTS?
↑
TO ORDERING


- WHAT ARE THE CONSTRAINTS?
↑
TO LATENCY


- WHAT ARE THE CONSTRAINTS?
↑
TO
GLOBAL
GUARANTEES

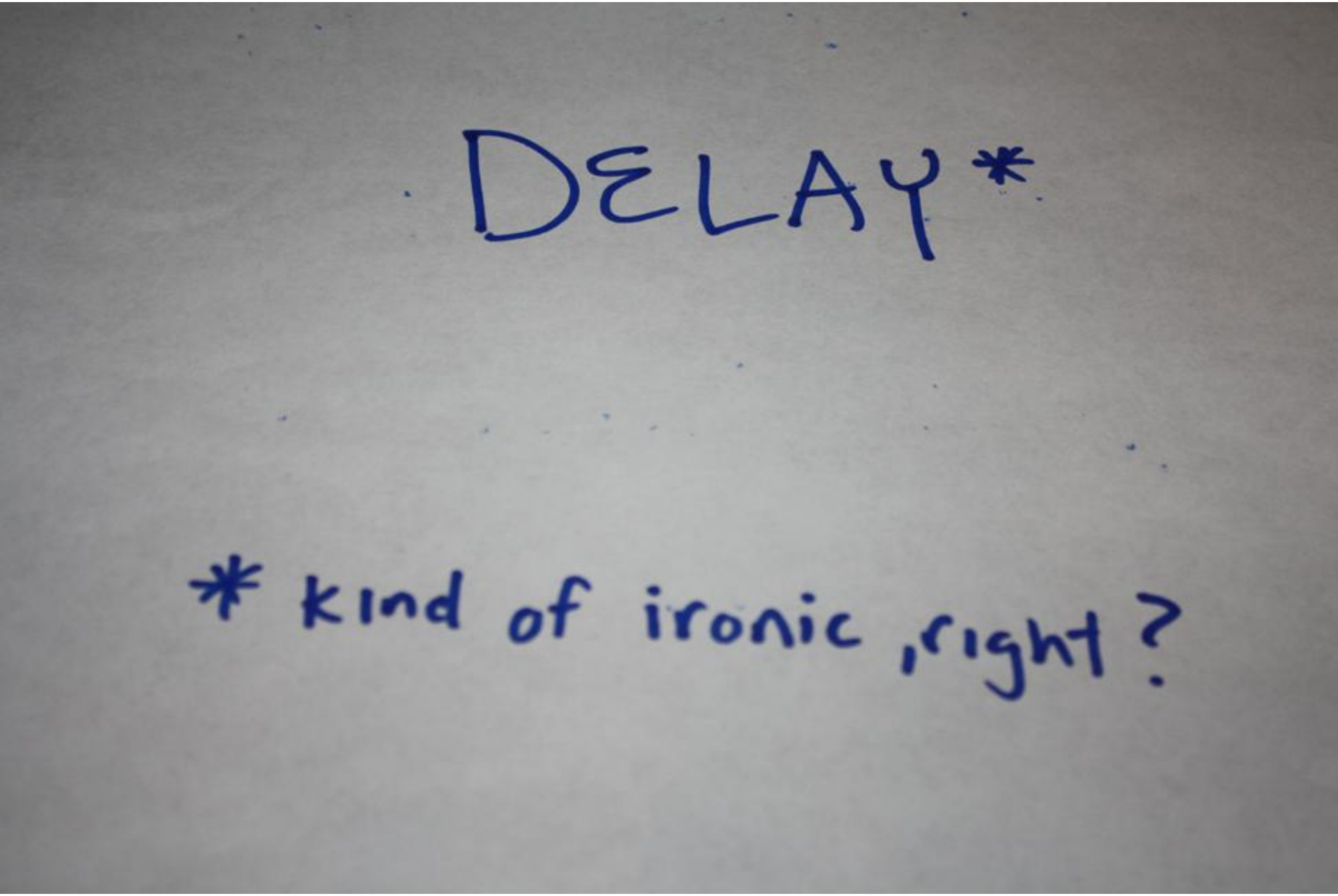

- WHAT ARE THE CONSTRAINTS?
↑
TO
LOCAL
GUARANTEES

THE SECRET

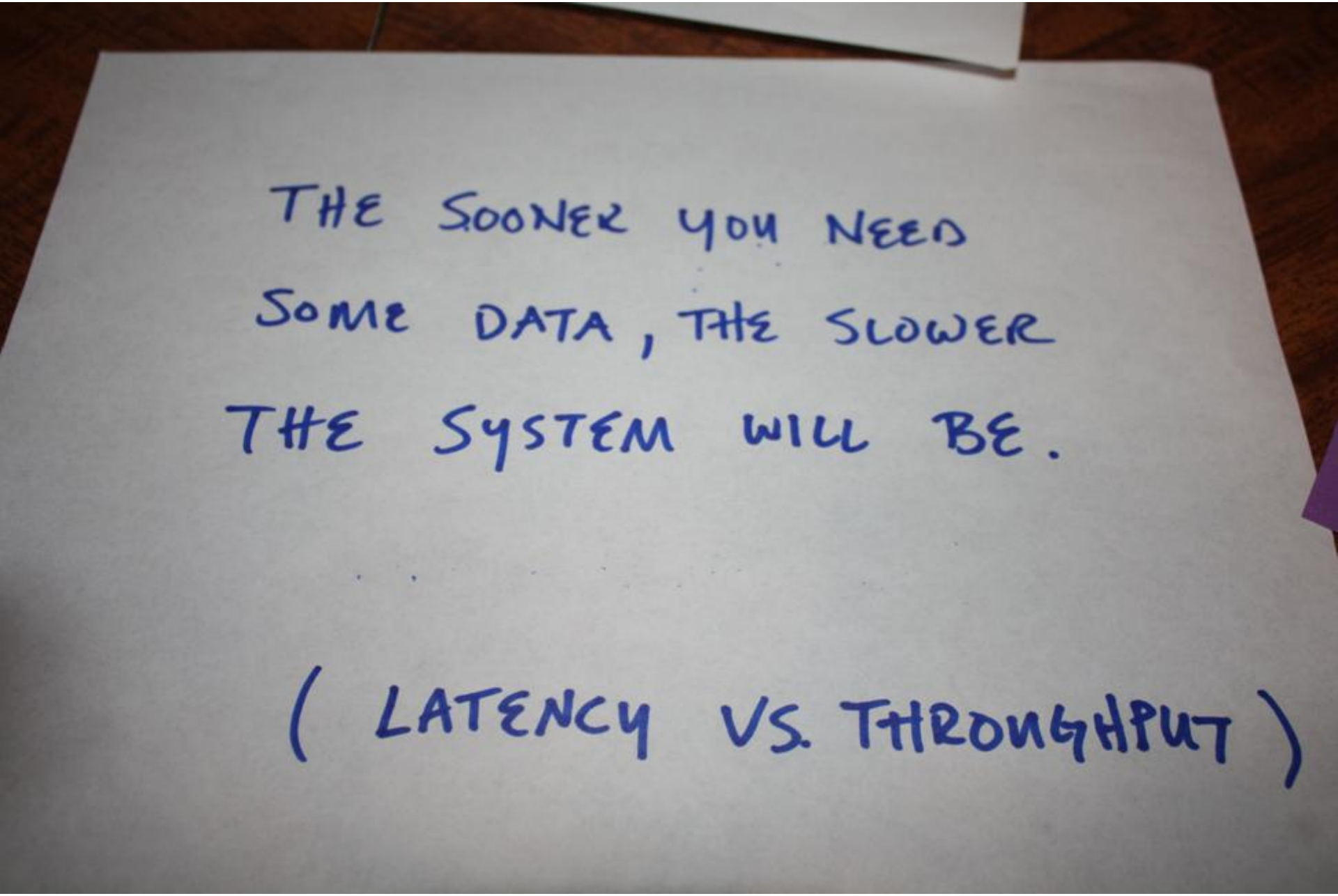
of OPTIMIZED

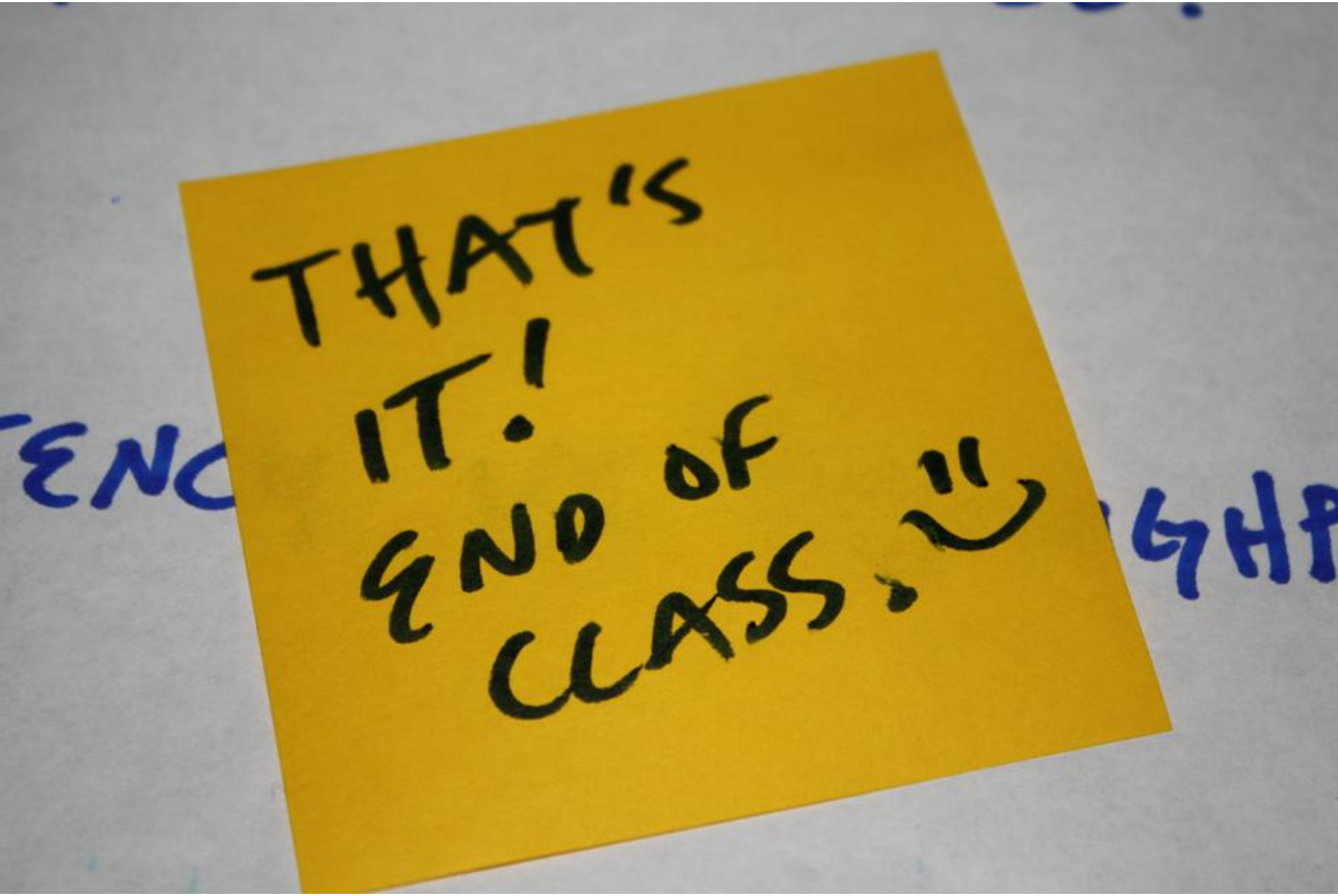
concurrent design

IS...


DELAY*
* kind of ironic, right?

THE SOONER YOU NEED
SOME DATA, THE SLOWER
THE SYSTEM WILL BE.


THE SOONER YOU NEED
SOME DATA, THE SLOWER
THE SYSTEM WILL BE.
(LATENCY VS. THROUGHPUT)


THAT'S
IT!
END OF
CLASS.