Midterm 2 Review

CS 165

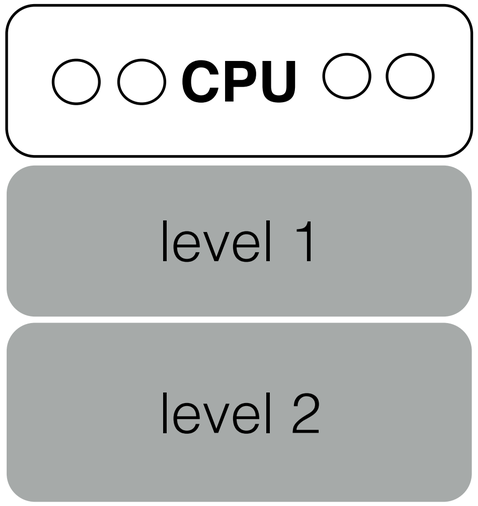
Topics after Midterm1

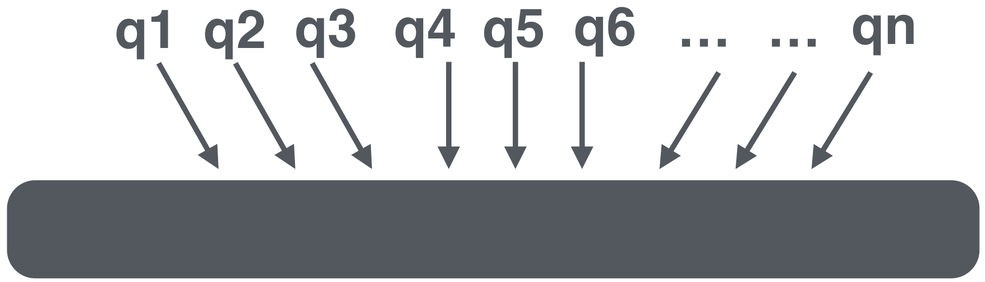
- Shared and fast scans
- Joins
- Updates

Shared Scans

N queries/selects in parallel on the same column
1) cost (L1 misses) for plain scan
2) devise shared scan approach
3) cost (L1 misses) for shared scan

corner cases:
what if queries do not arrive at the same time?
what if some queries are faster than others?
is there a limit to the number of queries in a shared a scan?





(assume simplified memory hierarchy)

Column >L1, Column < L2, L1 block = L2 block = block bytes, Column = C blocks
CPU can read directly from level 1 only

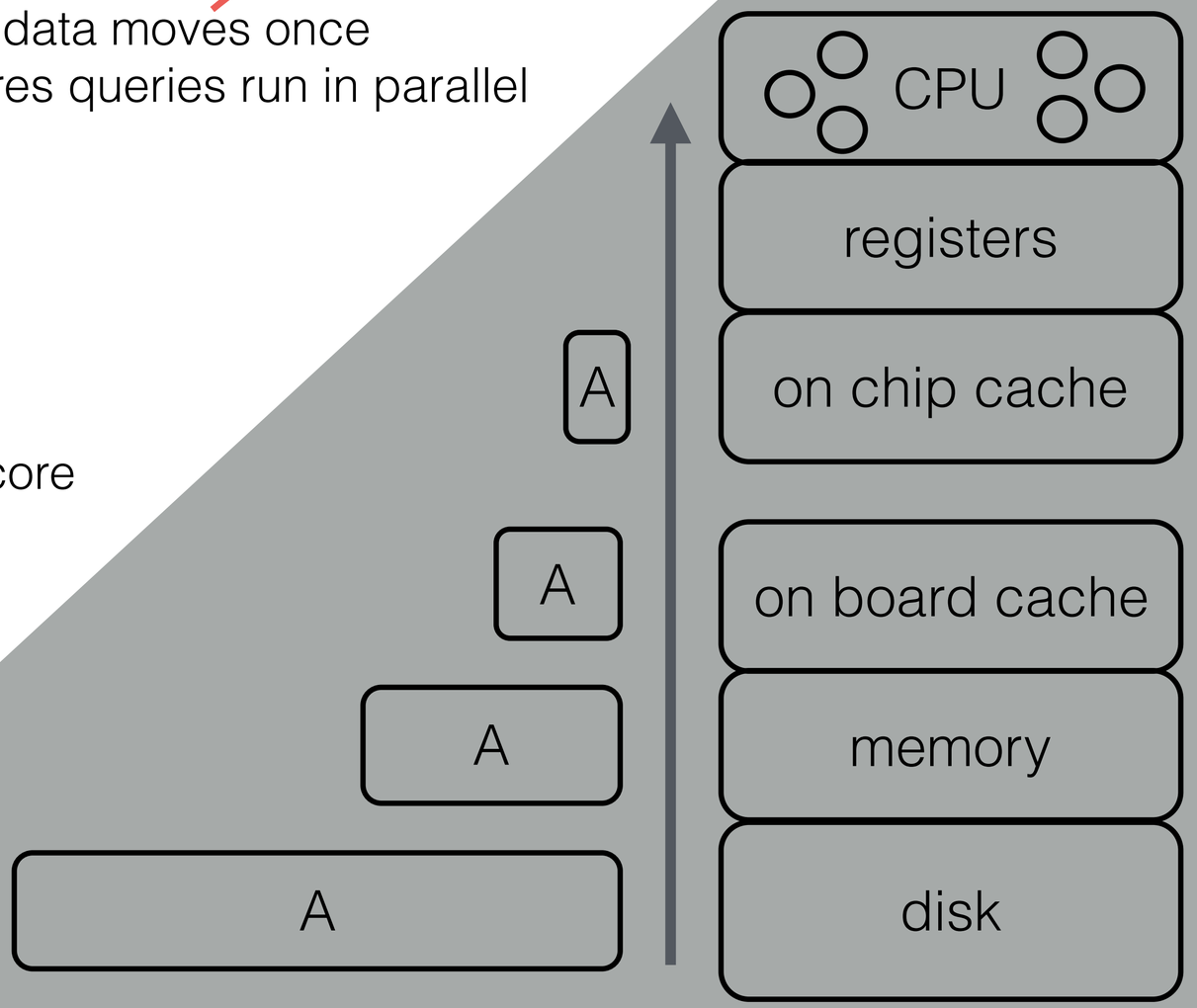
Remember the aim is
to reduce data movement

data moves once
# of cores queries run in parallel

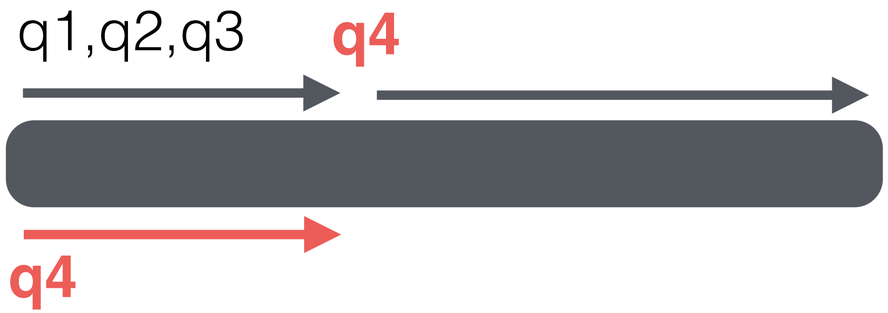
1) gather queries

2) schedule queries on same data to run in parallel

3) each query gets a thread/core from thread pool



Attach queries arriving asynchronously
elevate queries that are slow



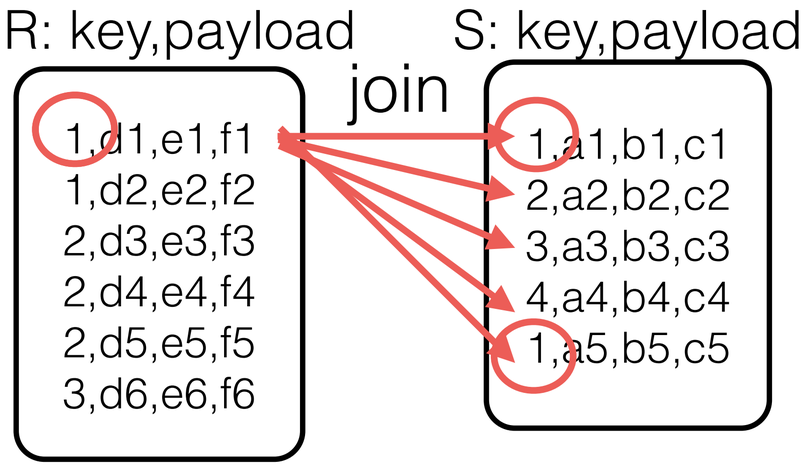
Is there a limit to the amount of queries that a shared scan system can handle ?

Observe that each of these queries have their own working set. Assume a scenario where the working set of multiple queries exceed the size of the cache, it will lead to eviction of some of the required data leading to increased cache misses.

Grace Hash Join

for every
L.key = R.key pair
return [L.pos,R.pos]

data/results
one column-at-a-time



CPU

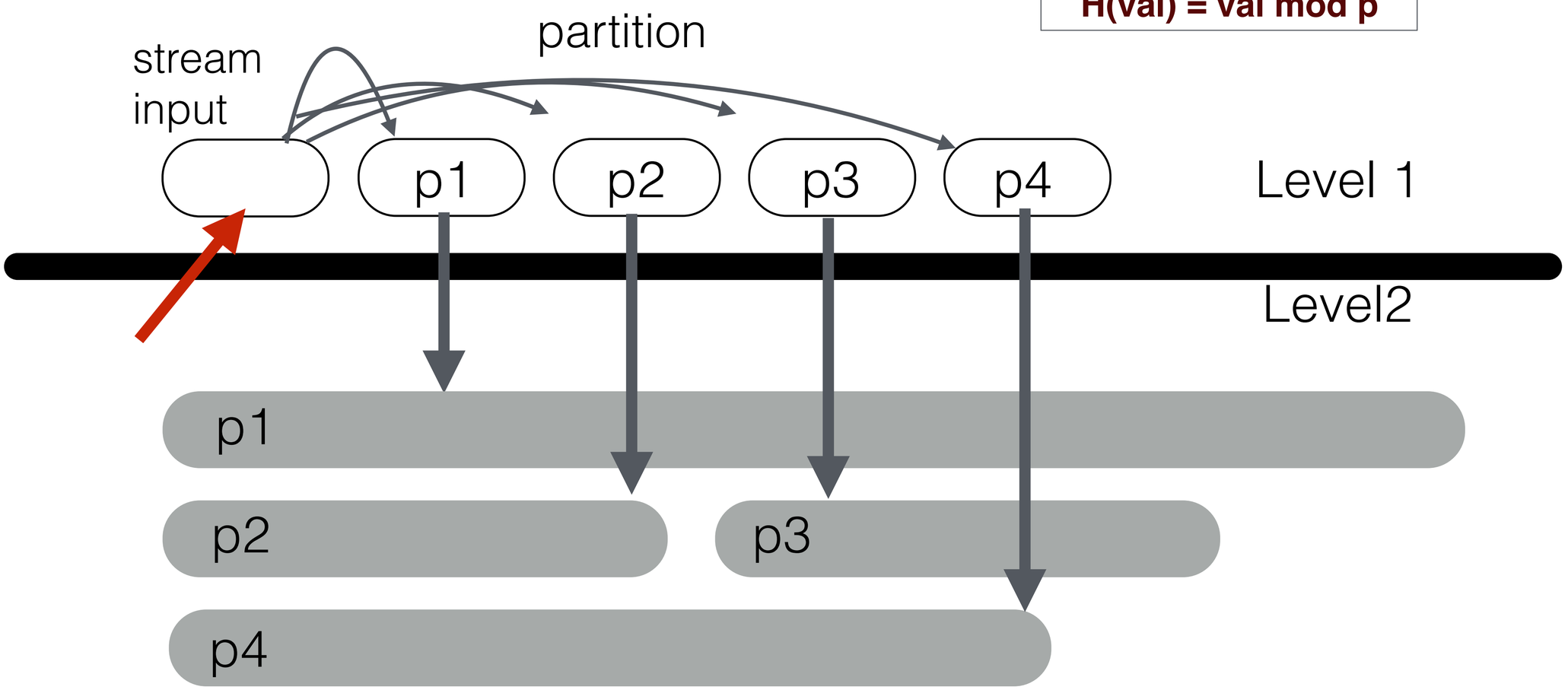
level 1

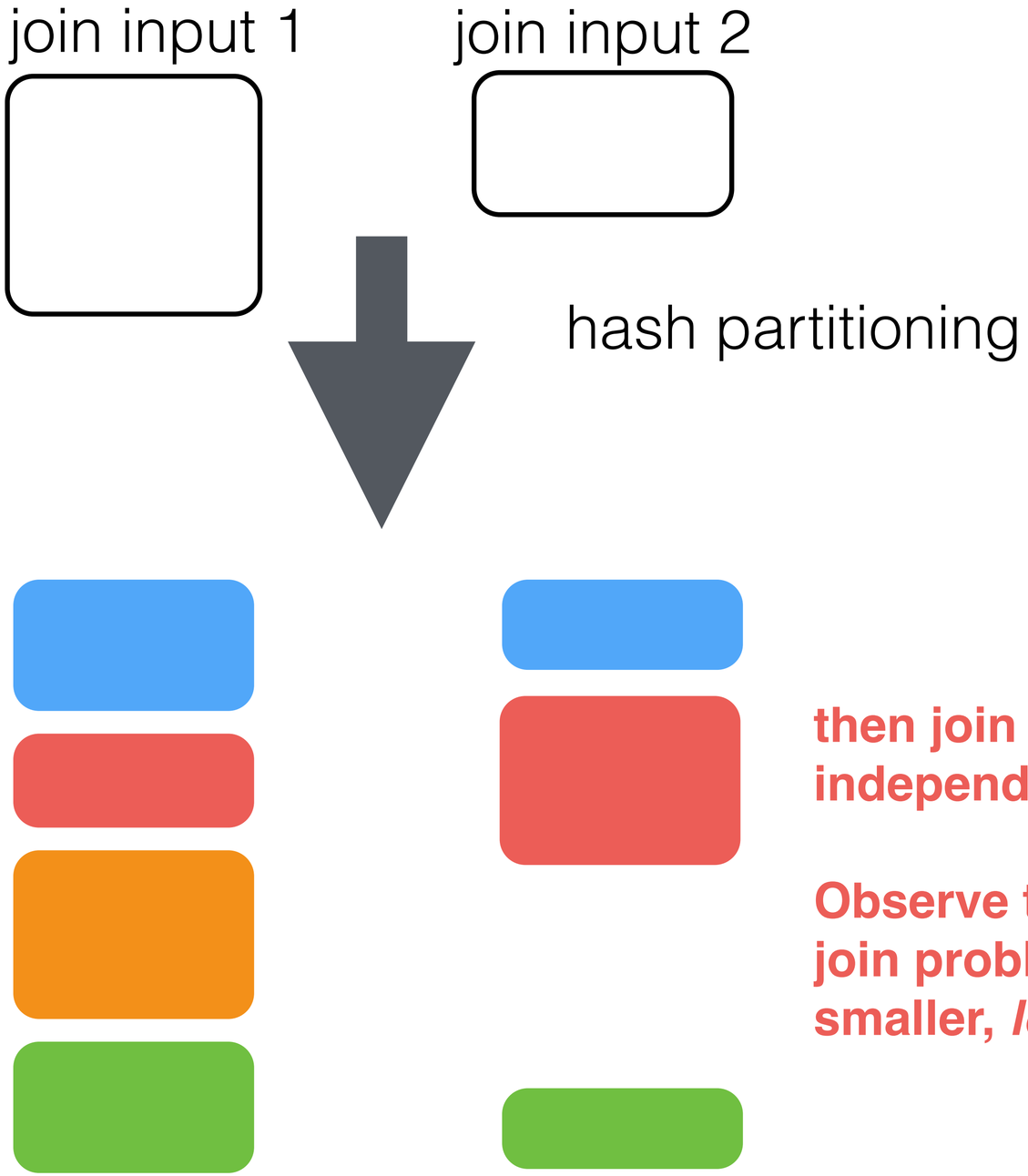
level 2

1) Design a grace join algorithm and give its cost
2) Which table do we start from?
3) What if partitions end up being > L1-2?
4) When can we do grace join in 2 passes max?
5) How can we utilize multi-cores?
6) Can you keep all cores at 100% all the time?

1. read input into stream buffer, hash and write to respective partition buffer
2. when input buffer is consumed, bring the next one
3. when a partition buffer is full, write to L2
we can partition into L1-1 pieces in one pass

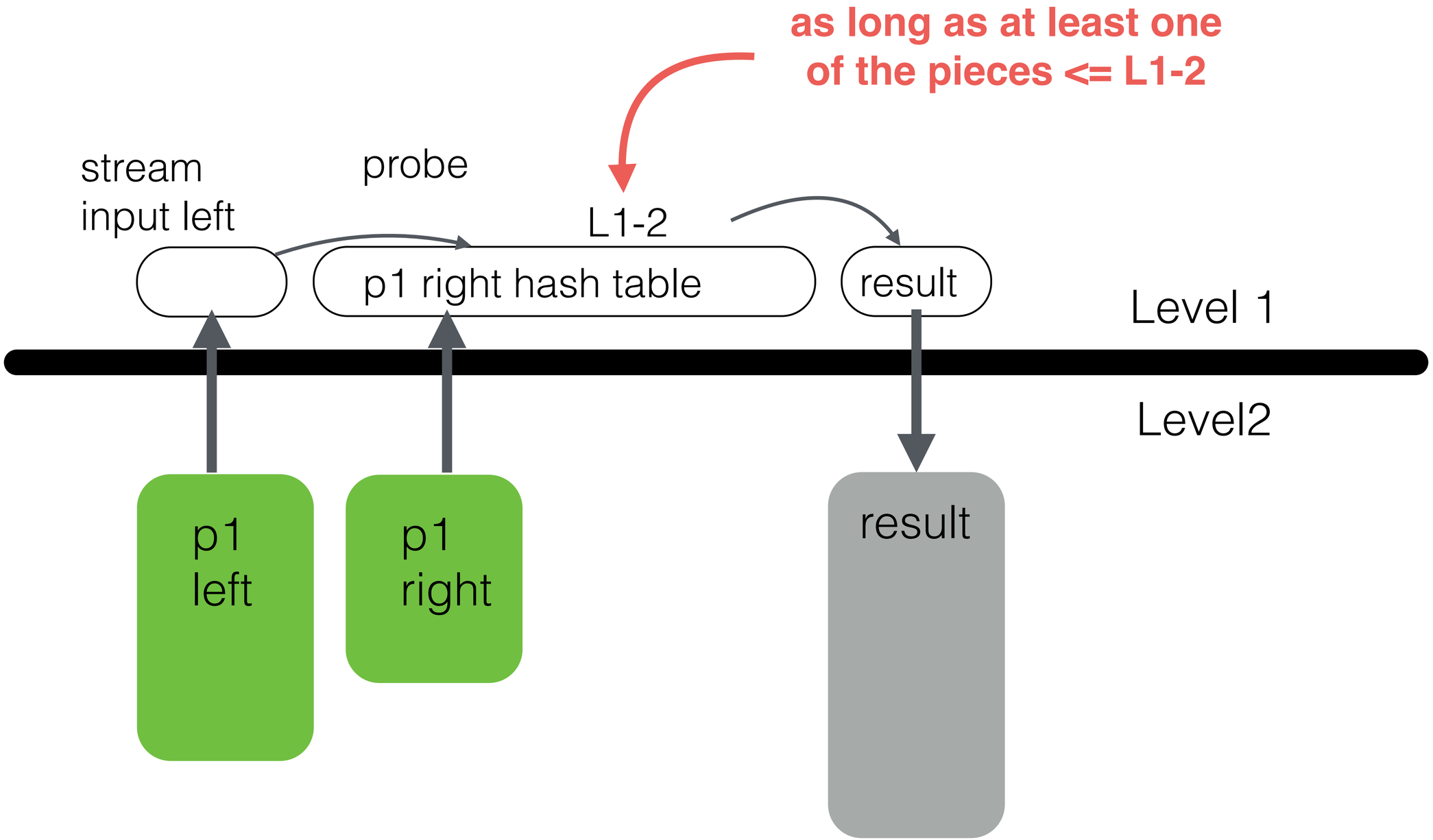
A simple hash function
H(val) = val mod p

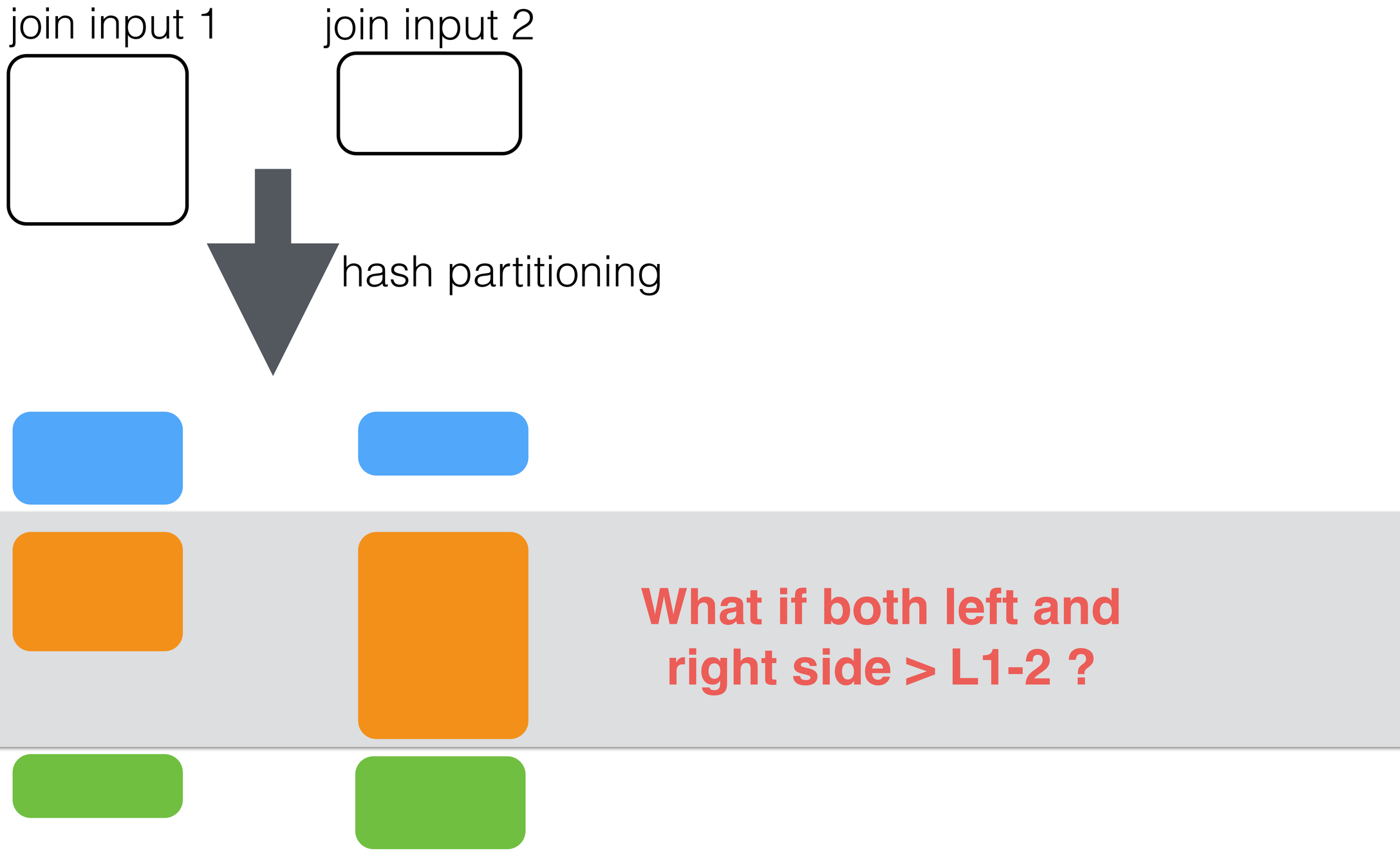


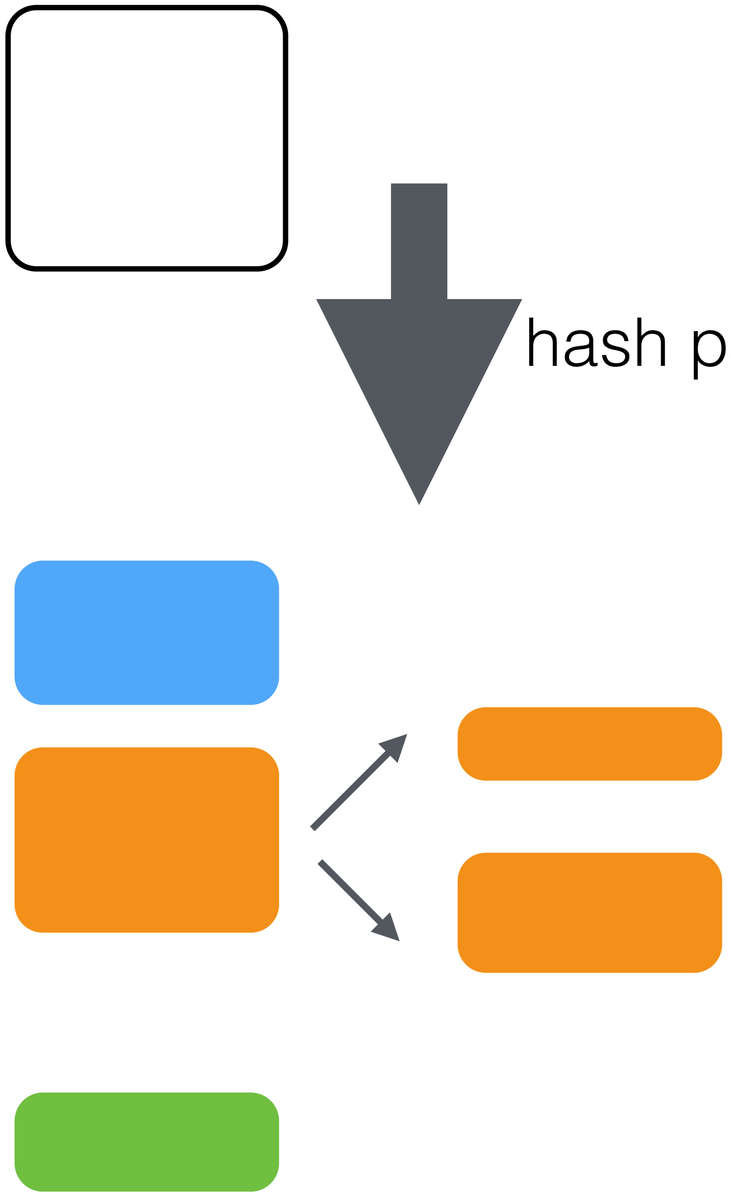


then join each pair of partitions independently in memory

Observe that we took the bigger join problem and divided it into smaller, *locally* solvable problems


as long as at least one
of the pieces <= L1-2
stream
input left
probe
L1-2
p1 right hash table
result
Level 1
Level2
p1
left
p1
right
result


join input 1
join input 2
hash partitioning
What if both left and
right side > L1-2 ?



apply recursively if a partition does not fit in memory

When do we have to repartition ?

1) At least one side should fit in L1-2, else we will have to repartition.

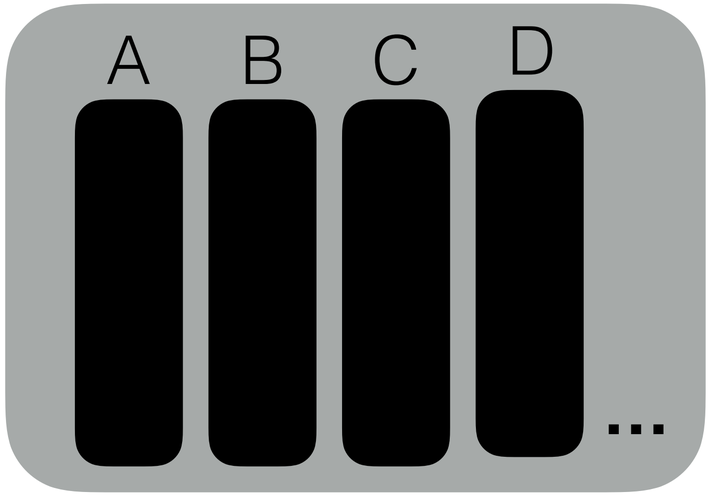
2) If all the partitions we create are such that at least one side of the partition fit in L1-2, we are good.

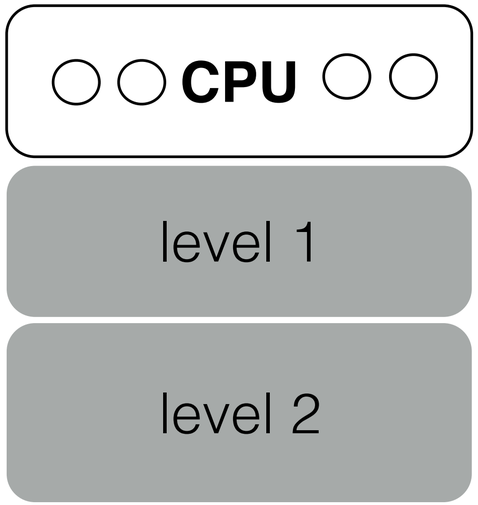
3) The maximum number of partitions we can create in one pass is L1-1

4) So if R/L1-1 <= L1-2, this implies that R<=(L1-1)(L1-2), we will not have to repartition any pieces

Updates / WAL

update all rows
where A=v1 & B=v2
to (a=a/2,b=b/4,c=c-3,d=d+2)

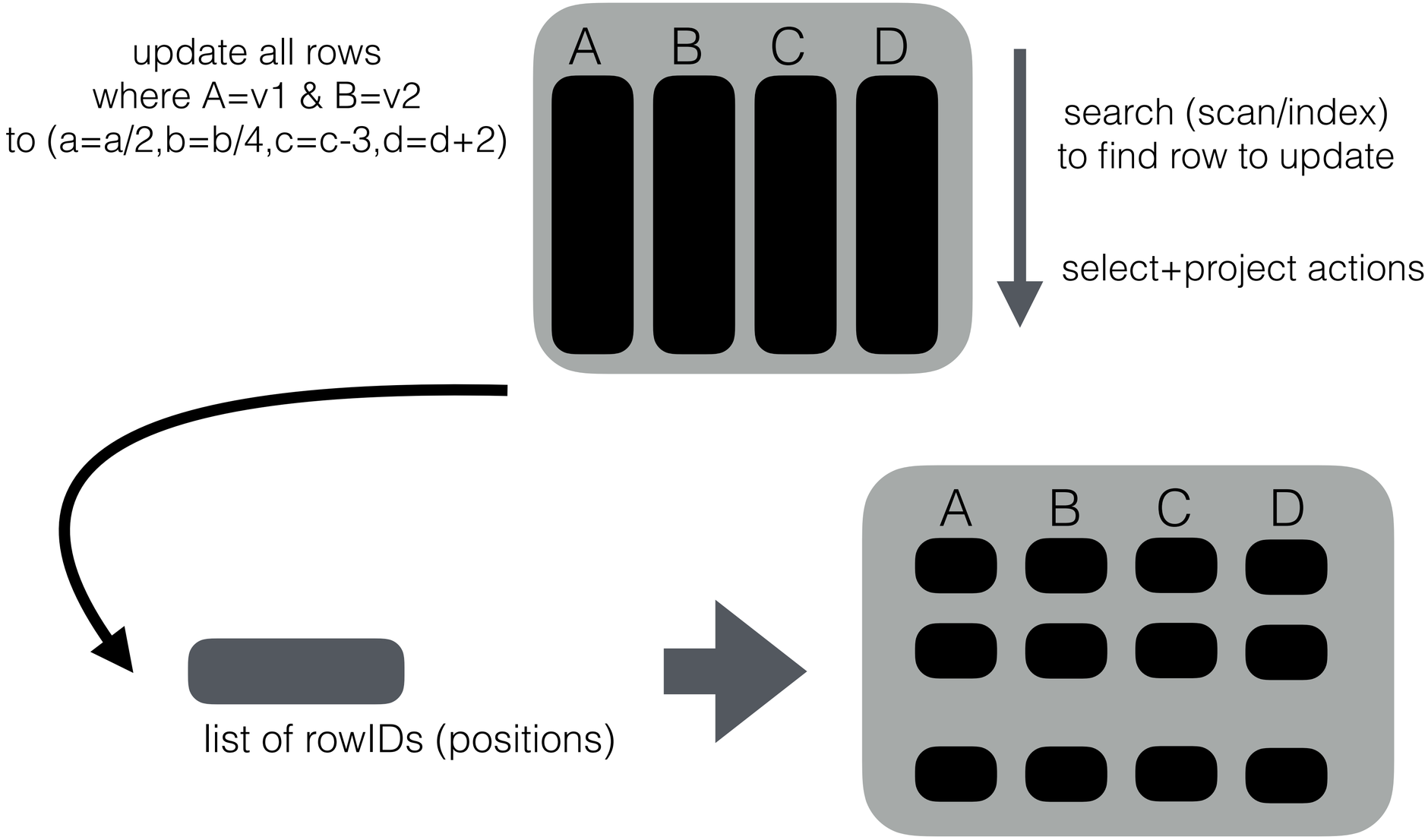




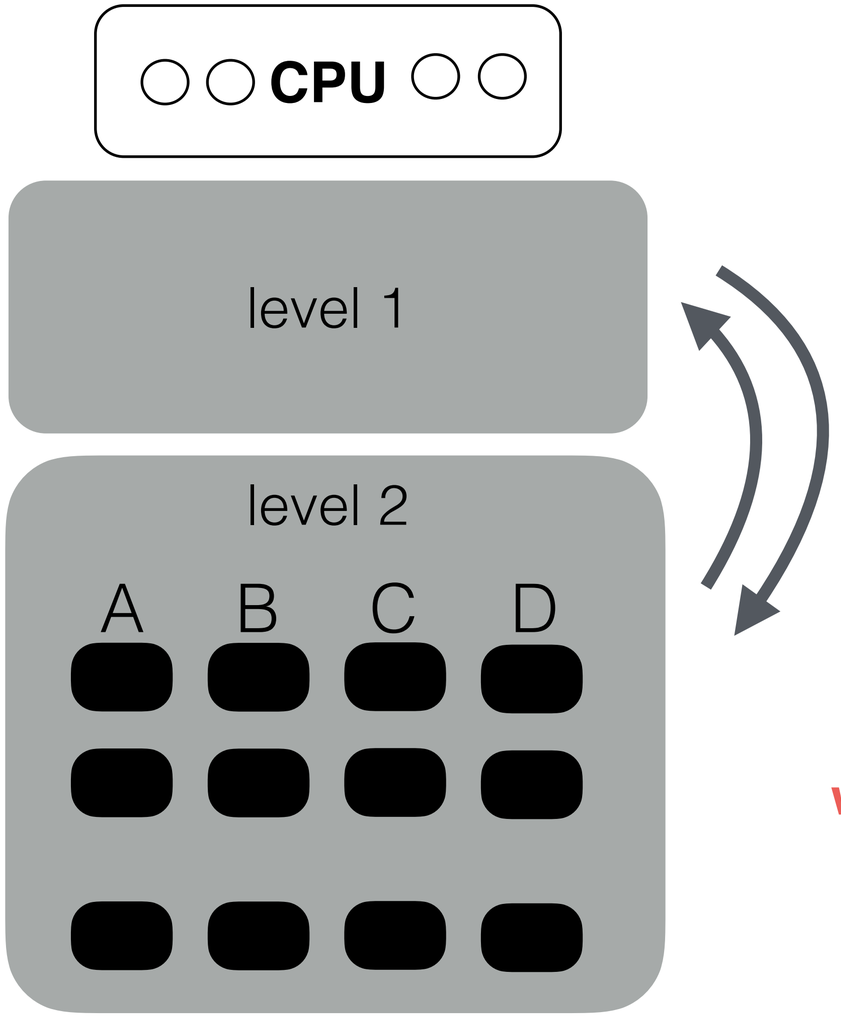
how to perform updates efficiently and correctly?
correctly=all or nothing

problems to worry about (?):
what if user/applications aborts?
what if power goes down?
what if there is an earthquake in our city?
what if aliens come to earth?

(assume simplified memory hierarchy)
all data fit in L2, not all data fit in L1
L2 is non-volatile, L1 is volatile



we know what to update but nothing happened yet



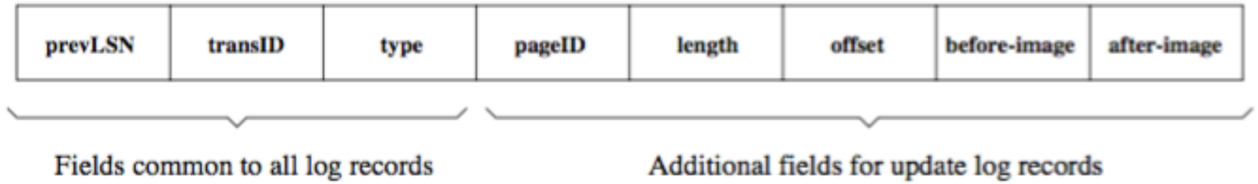
read page in L1
update
persist to L2

if problem (power/abort)
before we write all pages
we are left with an inconsistent state

WAL: keep persistent notes as we go so we can resume or undo

Write Ahead Logging

Modifications are written to a log before they are applied.



Modifications are *provisional* until a commit entry is written to the log.

On system crash:

1. Analyze the log.
2. Redo all the operations (after the last checkpoint).
3. Abort operations that were not committed.

Remember to:

Understand the set up of the database and the hardware.

Read the questions carefully.

Provide answers to each part of a question.

Try and attempt as much as possible.

Always provide reasons.

Write legibly.