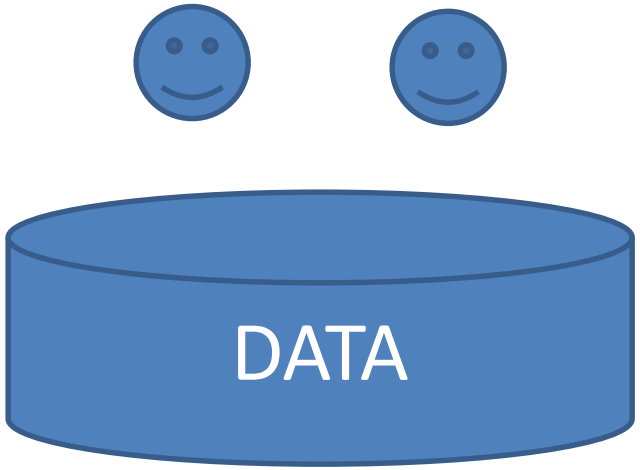
REPLICATED DICTIONARY AND LOG

Replicated dictionary problem

- Efficient solutions to the replicated log and dictionary problems. Wuu and Bernstein PODC 84
- Replication is a fundamental method for fault-tolerance.
- Replication is also used for performance.
- Dictionary is a common data structure which is often fully replicated on all sites.
- (Blockchains are also fully replicated)

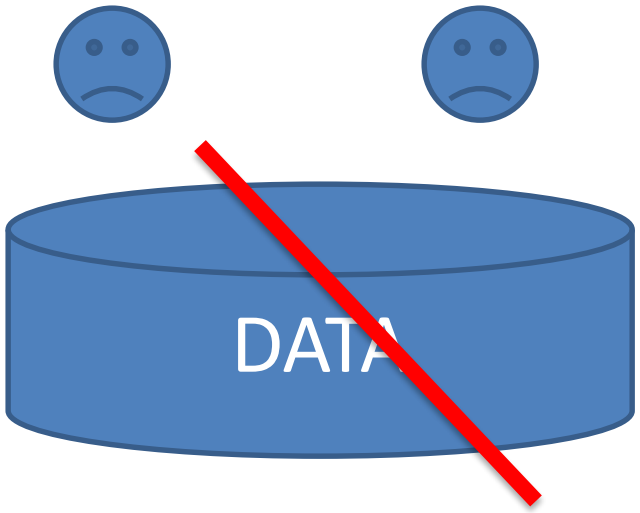
Replication motivation

- What if data is stored in ONE place?



Replication motivation

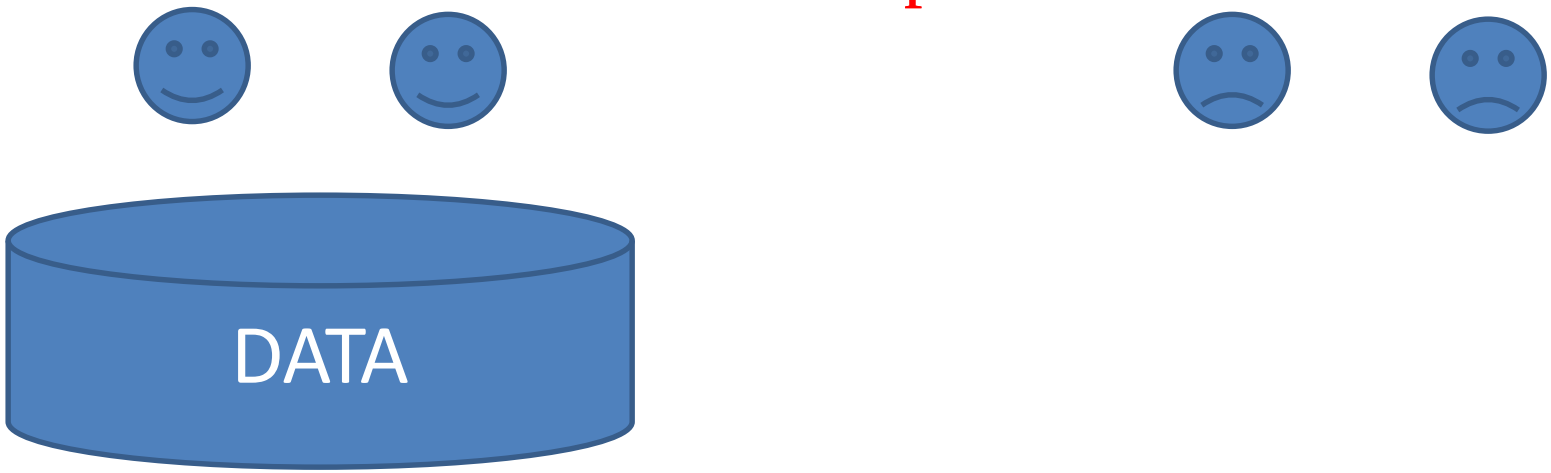
- What if data is stored in ONE place?
  - Failures cause data to be inaccessible (single point of failure)



CRASH!!

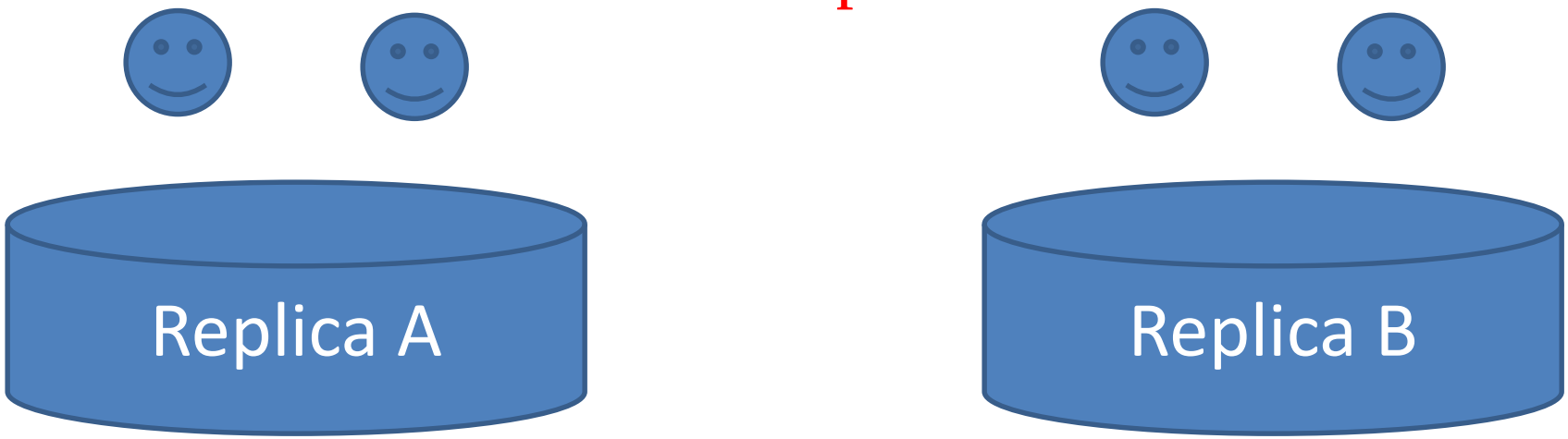
Replication motivation

- What if data is stored in ONE place?
  - Failures cause data to be inaccessible (single point of failure)
  - Users far away experience large latency to access data



Replication

- Replication requires maintaining data in multiple locations
  - Identical copies?



Basic assumptions

- Very general: Sites may crash, links may fail, partitioning.
- Asynchronous System
- Each site maintains a local clock (a counter).
- Use Lamport's event execution model and happens-before relation.

Motivating example for causality

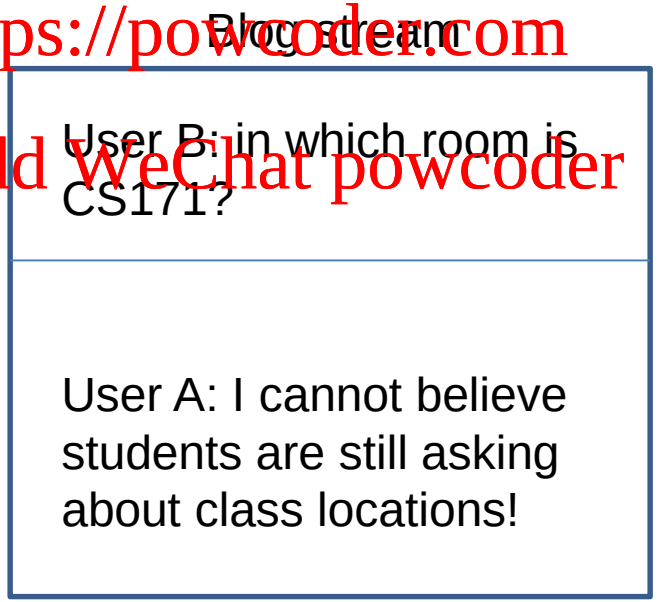
Micro-blogging application

Causality in micro-blogging

- A micro-blogging website has users and a micro-blog
- One global stream of users blog
- In what order should the stream display blogs?
- In a non-replicated deployment, order by time of arrival.

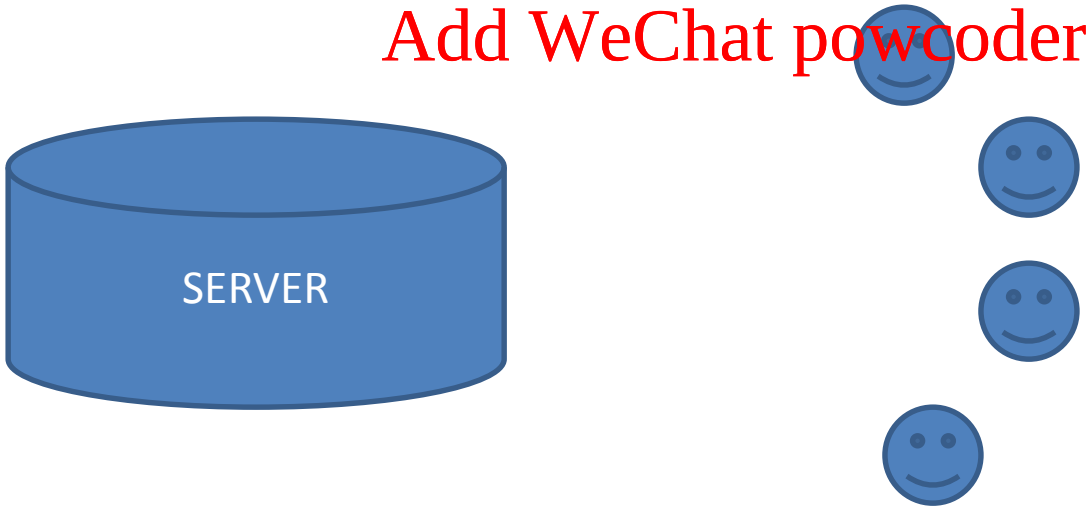
Causality between blogs

- A user A blogs blog a1 as a consequence of an earlier blog b2 (or blogs) by another user (users) B.



Causality between blogs

- This causality is trivially captured in non-replicated systems. User A made a comment after seeing what caused it (blog by User B) in the stream, which means it already exists.

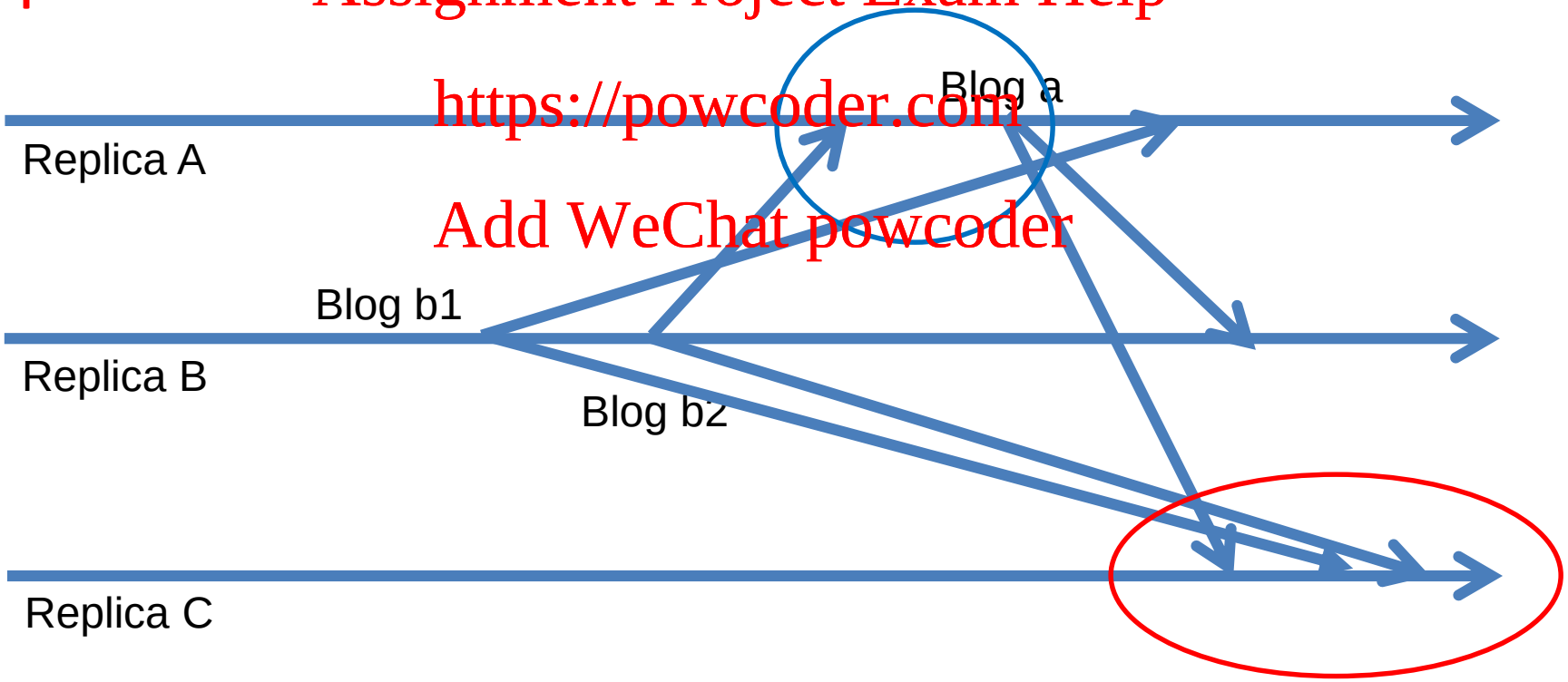


Causality between blogs

- For three replicas, A, B, and C. If blogs are just replicated asynchronously it will not guarantee causality.
- Naïve-replication:
  - When a blog is posted send to all replicas.
  - When a replica receives a blog, append to log.

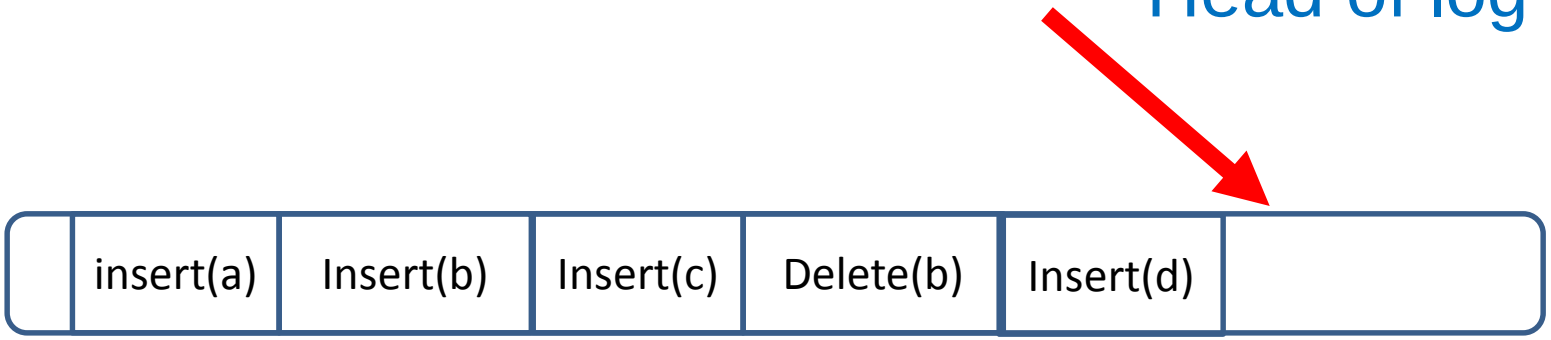
Causality between blogs

- Blog a is caused by blog b2,
- at replica C blog a is ordered before blog b2;
- problem!

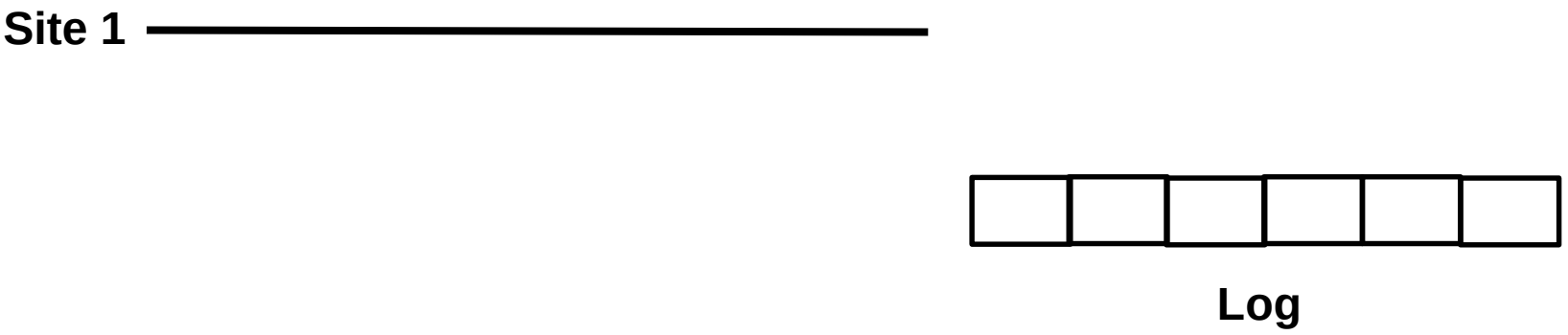


Log stores

- An append operation inserts a record at the head of the log
- Changing the content of a record: delete record and then insert
- Reading log => all log content.



The Log- Example



The Log- Example

- I(x)

Site 1

Log

The Log- Example

- I(y)

Site 1

Log

The Log- Example

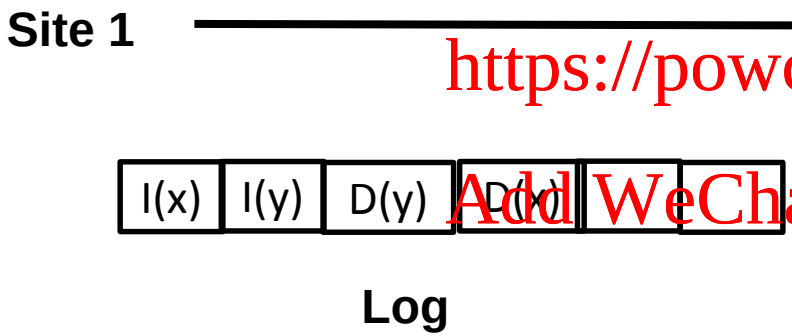
- D(y)

Site 1

Log

The Log- Example

- D(x)

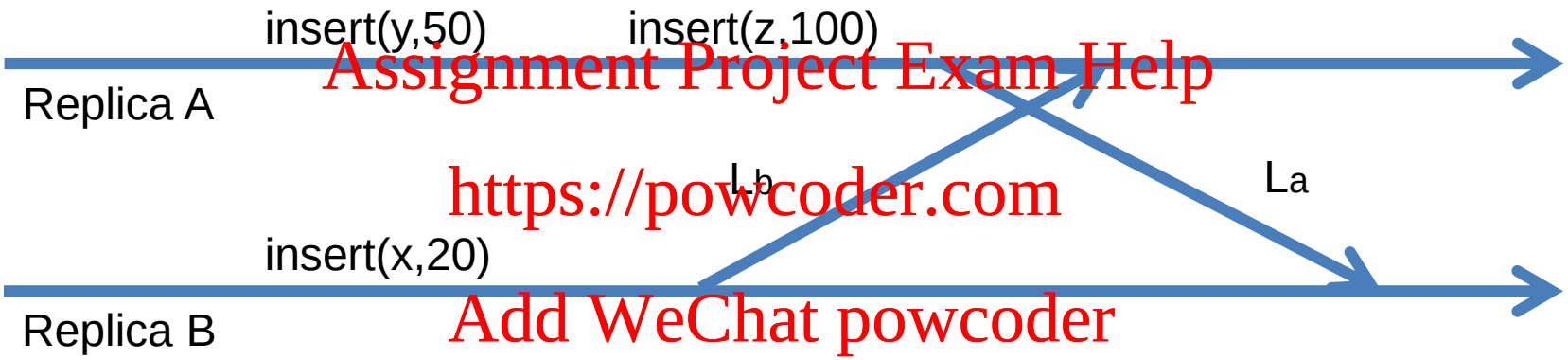


The log problem

- Each site maintains a copy of the log.
- The log contains local events, i.e.,
  - insert
  - delete
- The goal of the algorithm is to keep all copies of the log up to date.
- L_i is the copy of the log at site i.
- L(e) is the contents of log $L_{node(e)}$ immediately after event *e* is executed.

An example of a log

- Logs are transmitted between replicas



- What to include in the transmitted log?
- Lb = {insert(x,20)}
- La = {insert(y,50),insert(z,100)}

The log problem

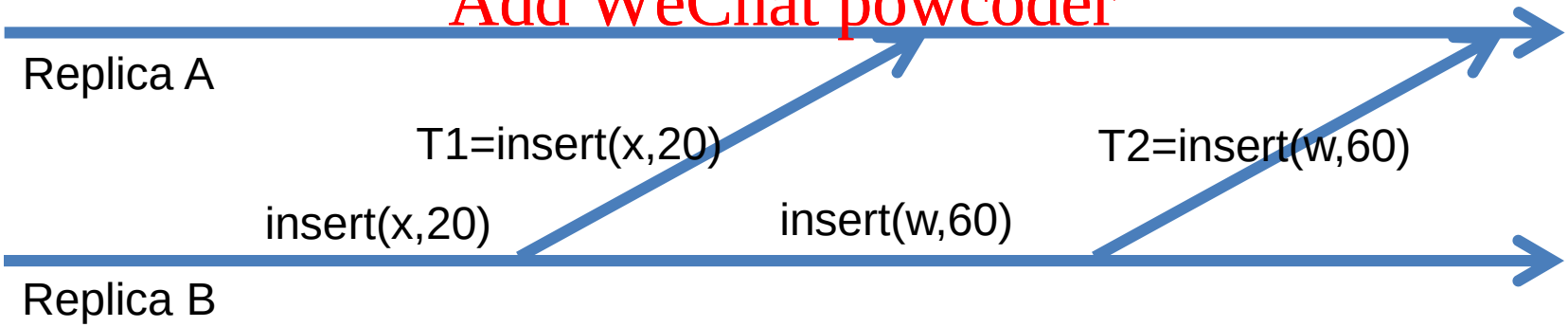
- Log Problem: find an algorithm that maintains the log such that given an execution <E, ⟶ >,

∀ events e,f: if f ⟶ e then f is in L(e)

- General approach:
  - For each local event, in the local log.
  - Exchange logs to update other sites
- Main question: when to exchange logs? With application communication to capture the happens before relation.

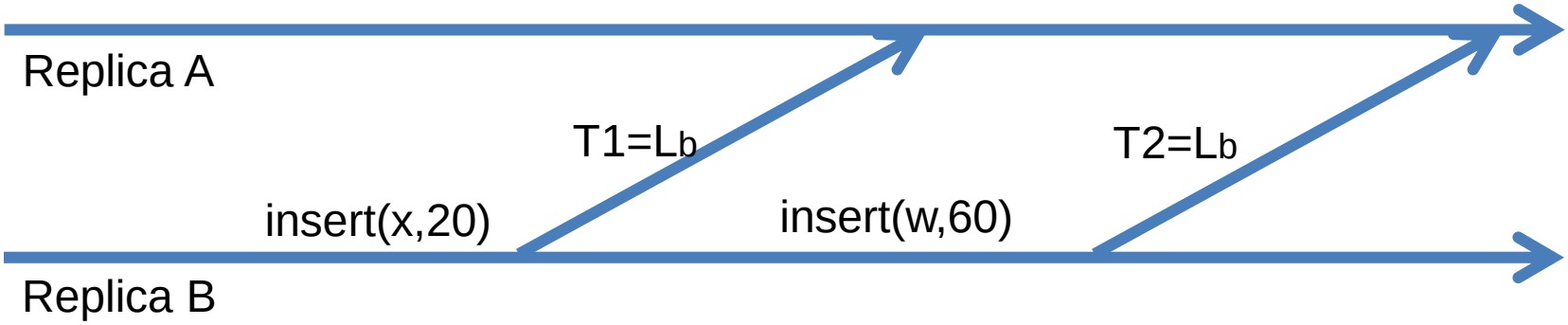
Attempt at Solution 1

- A solution:
  - Site i sends to site j all records in the log that were inserted since i last sent a message to j
  - WHY INCORRECT?
    - Log message lost – log messages reordered



Attempt at Solution 2

- Another solution:
  - each site i includes its log L_i with each message.
  - On receiving a message, a site j incorporates all new event records.
  - **BAD**:
    - Entire log sent with each message
    - Entire log kept at each node.

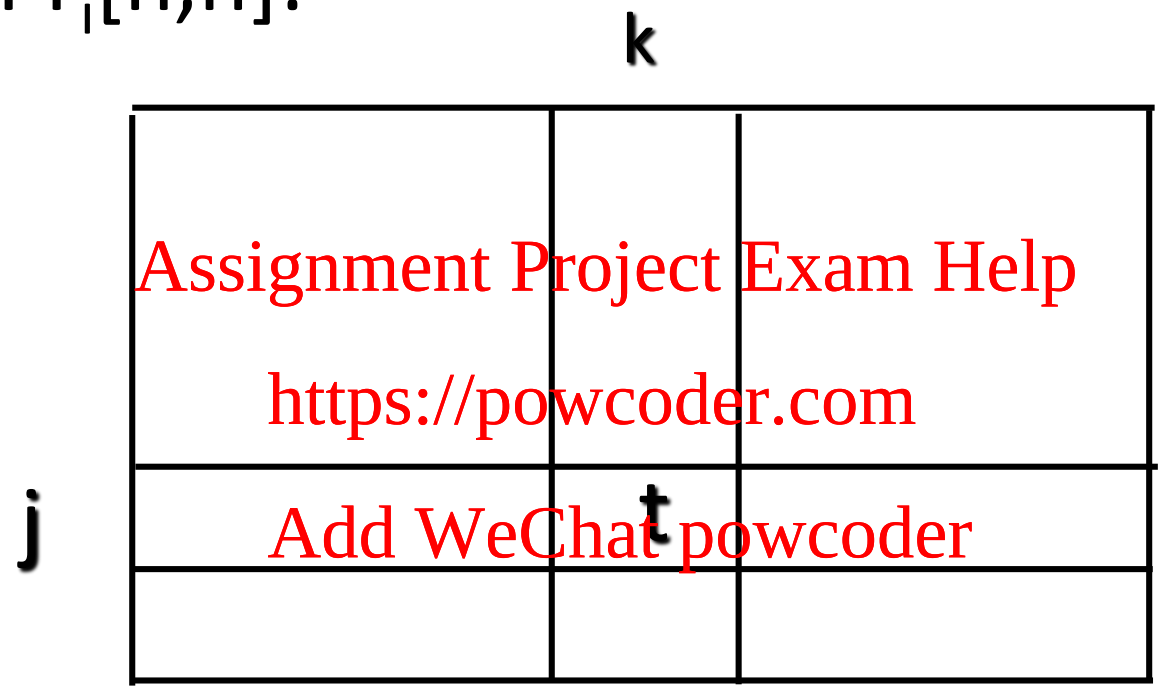


Efficient solution for log problem

- Observation 1: Once i "knows" that j "knows" of an event *e* (which may have occurred on site k), then i does not need to include event *e* in messages to j.
- Observation 2: Once i "knows" that **all** sites "know" about an event *e*, then i does not need to keep a record of *e* in its local log.

2 Dimensional Time-Table

- $TT_i[n,n]$:



- if $TT_i[j,k] = t$,

  then site i knows that site j has learned of all events that occurred at site k up to time t.

The 2 dimensional timetable

- site j might actually know about more events, but site i may not be aware of it.

- $TT_i[i,i]$ is the value of clock at site i.

- $TT_i[i,k]$ is the value of clock at site k of the most recent event at site k that site i is aware of.

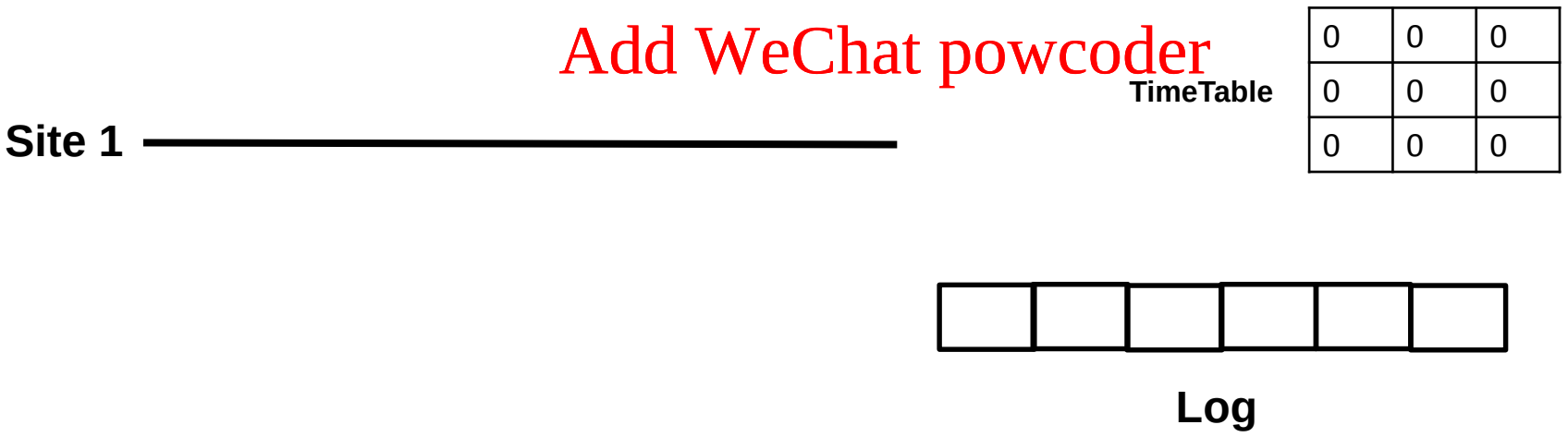
Two dimensional timetable

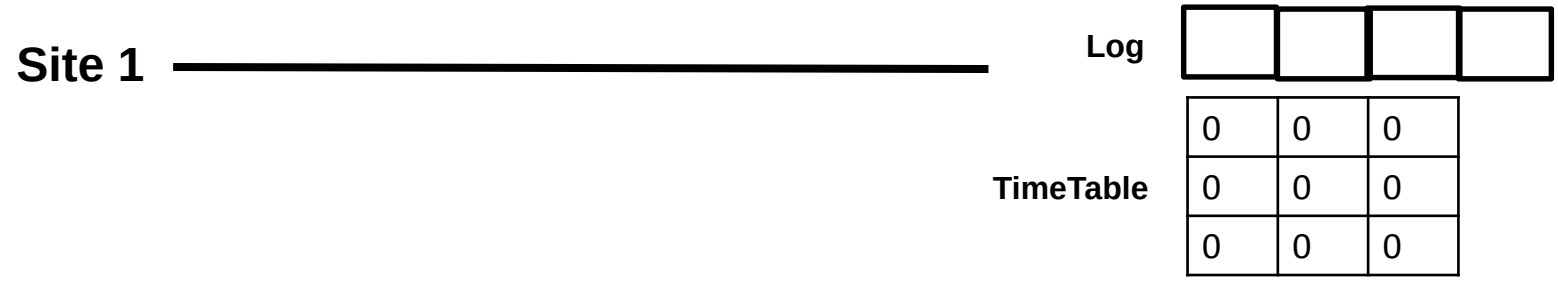
- Let hasrec(TT_i, e, k) be true iff

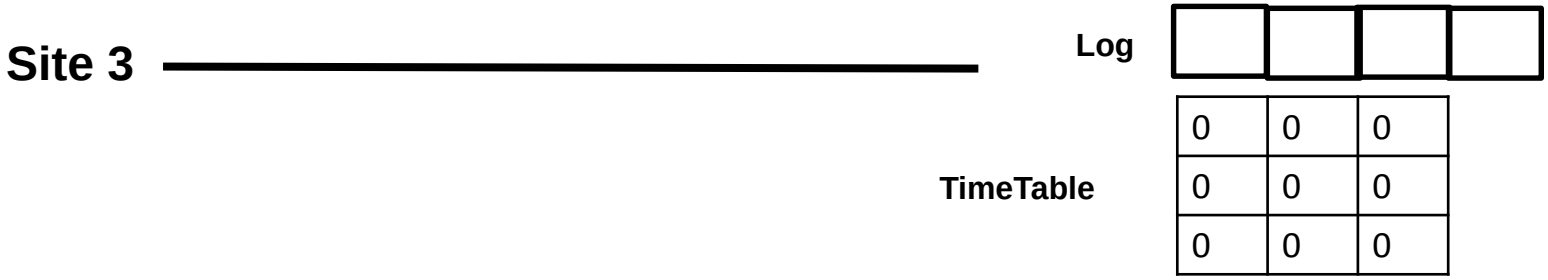
$TT_i[k, node(e)] >= time(e)$

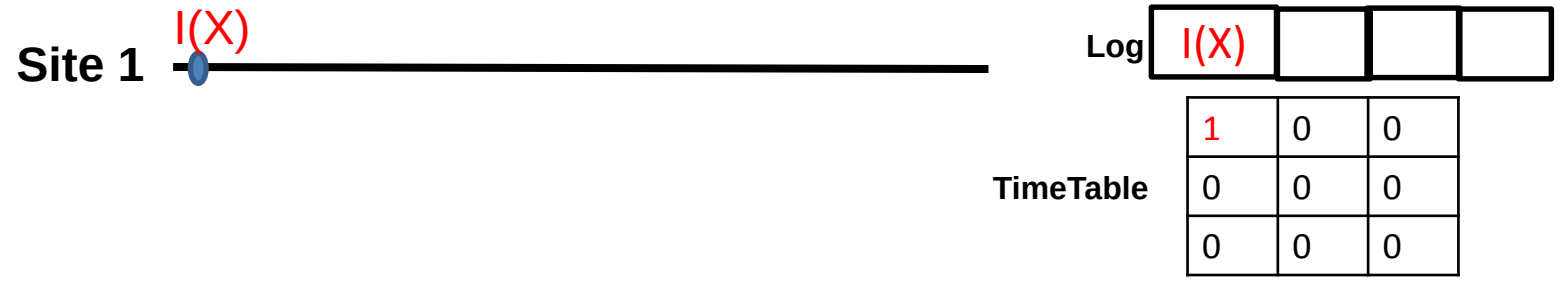
- The algorithm must guarantee that if hasrec(TT_i, e, k) is true, then site k has learned of event e.
- Note: site i need not send a record of event e to site k if hasrec(TT_i, e, k) is true.

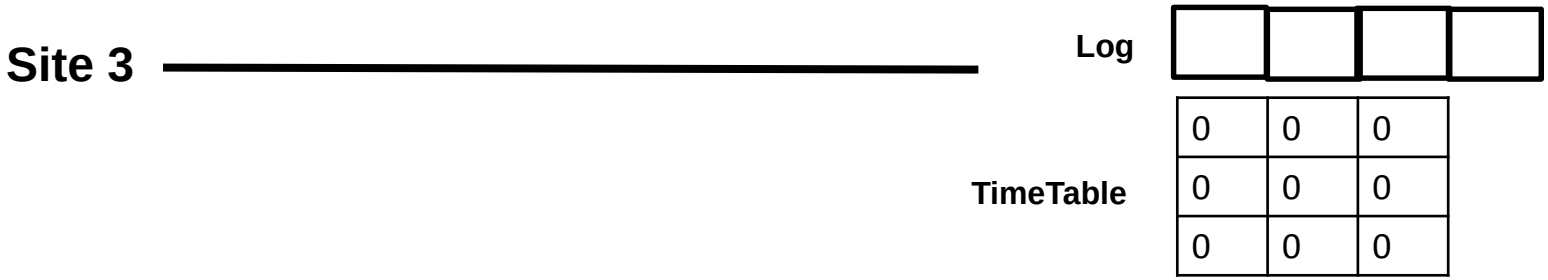
The Replicated Log- Example

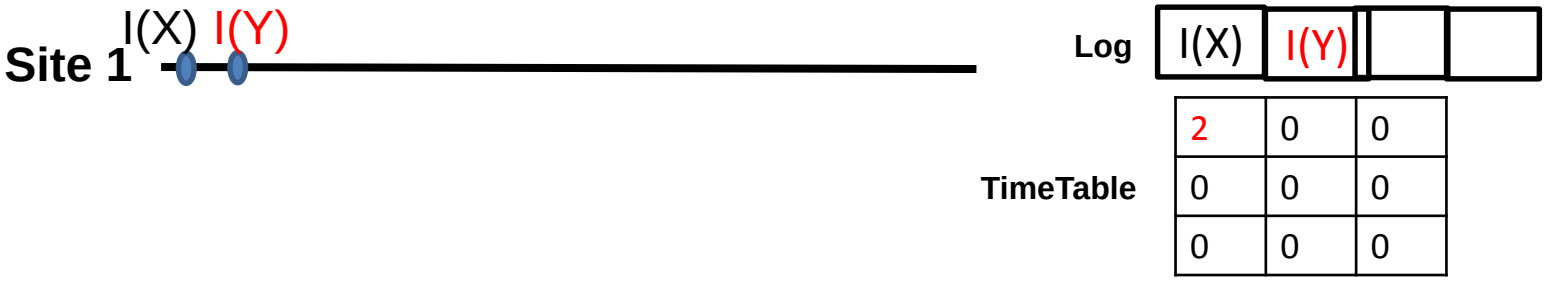


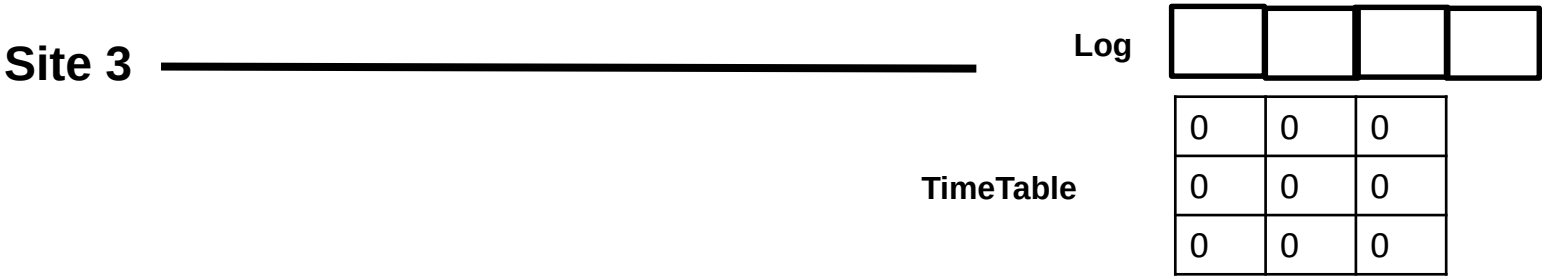


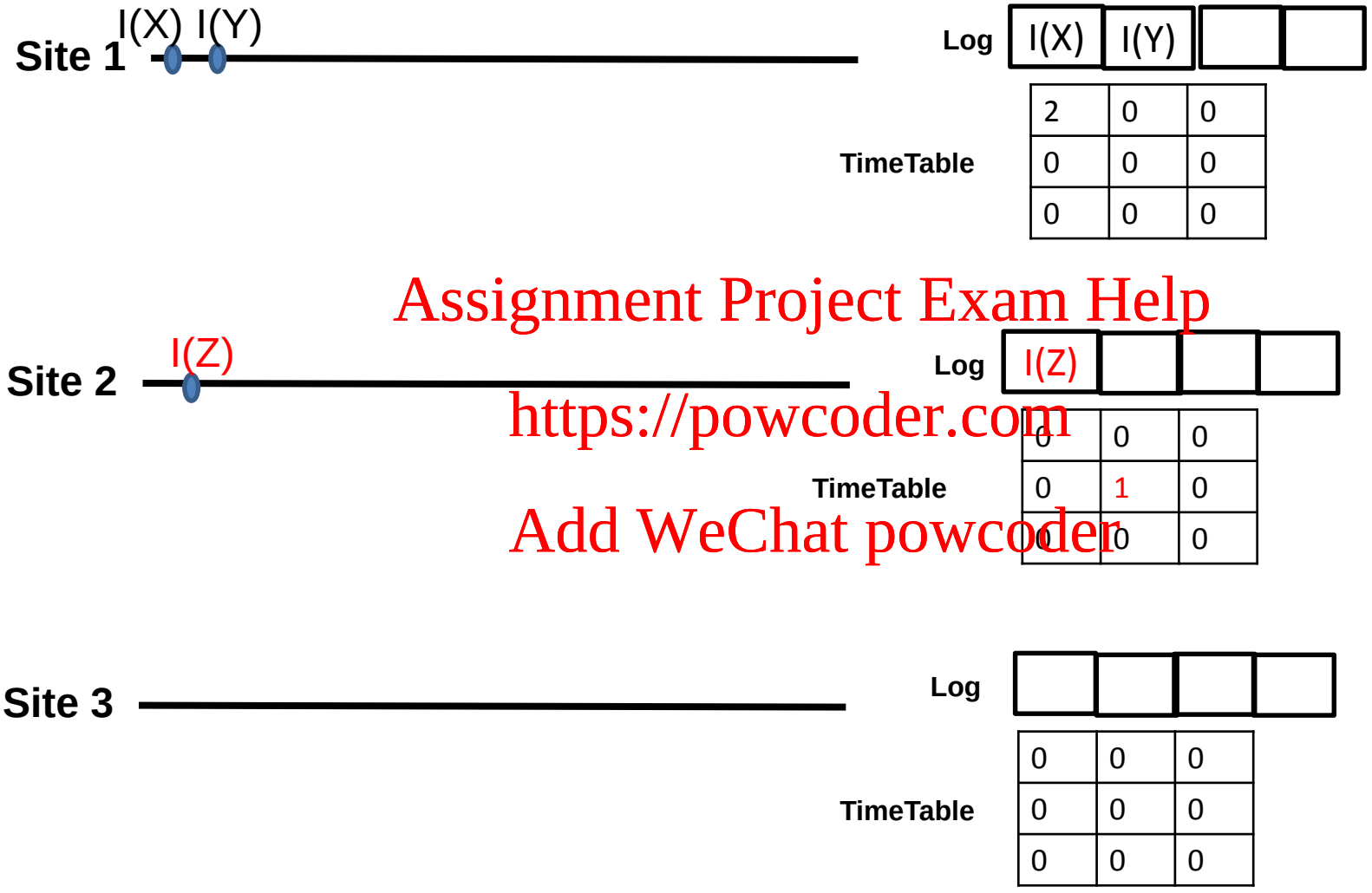


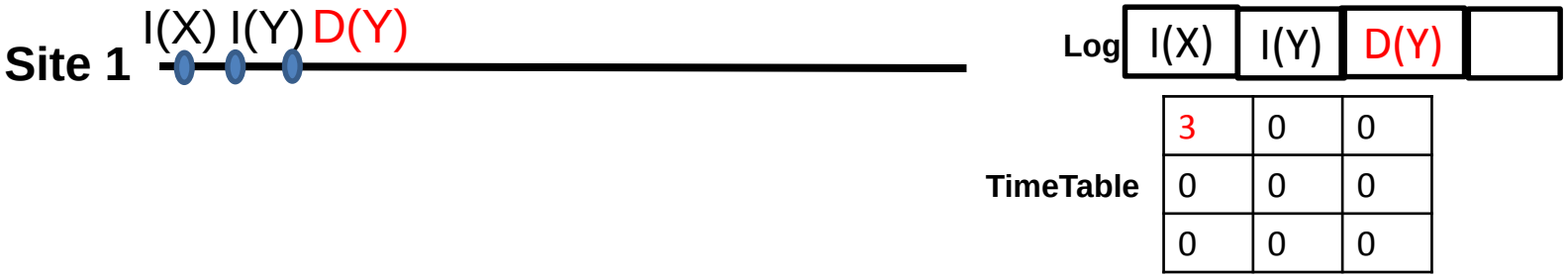


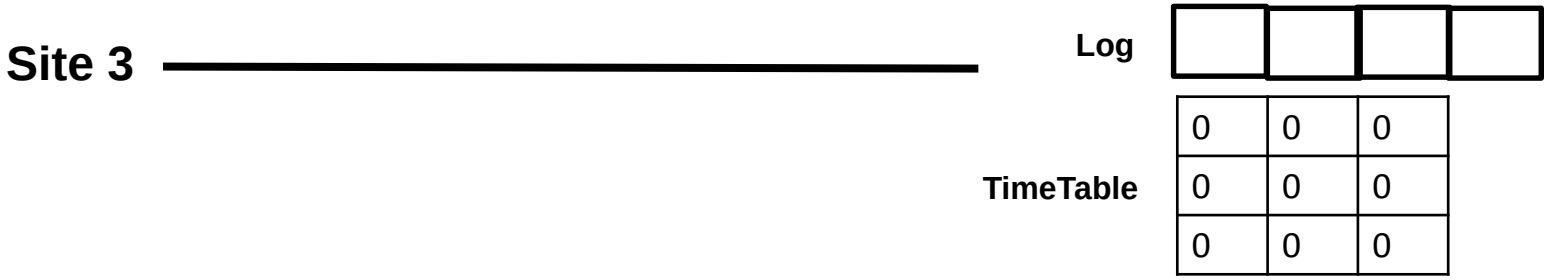


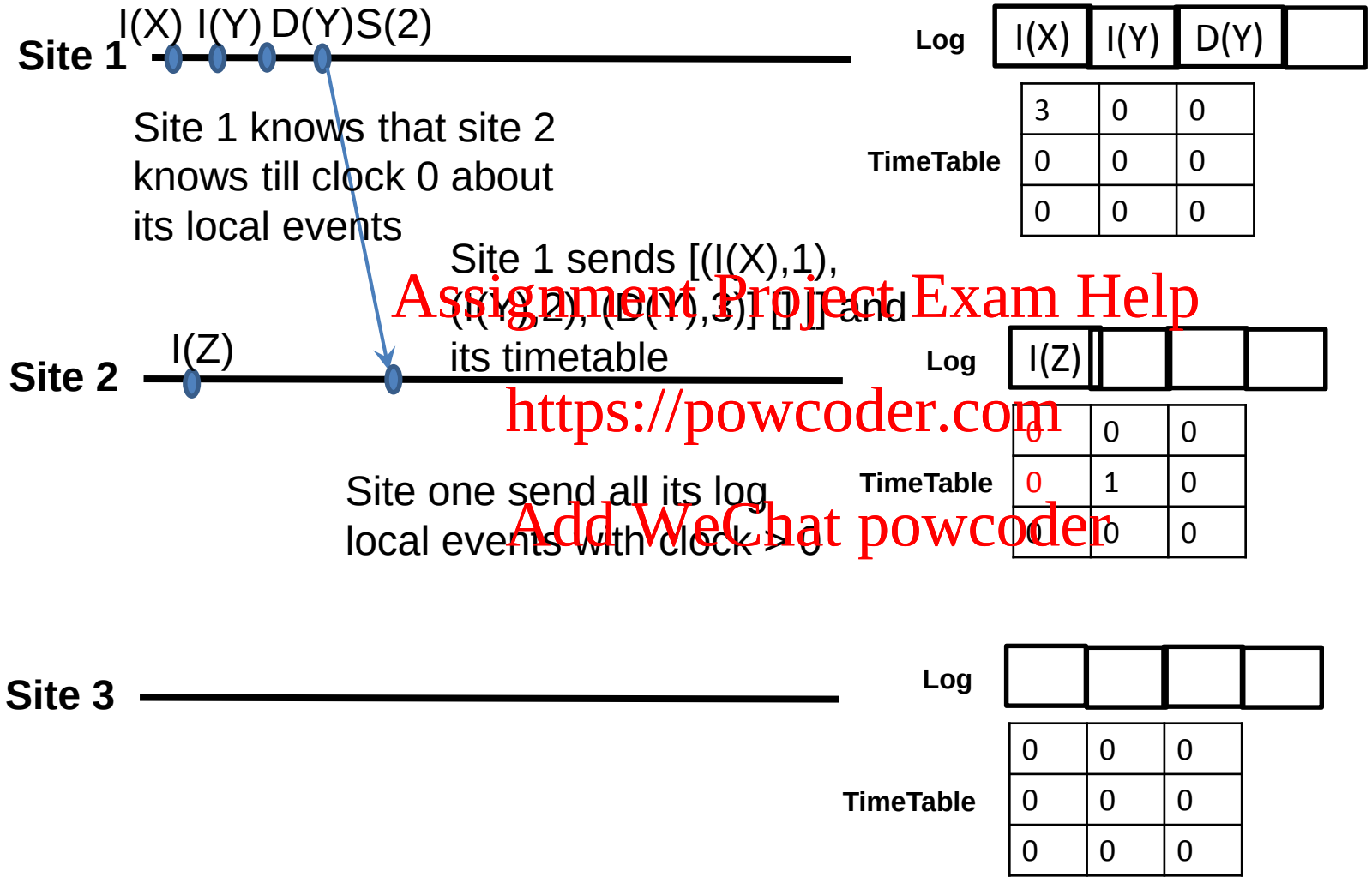




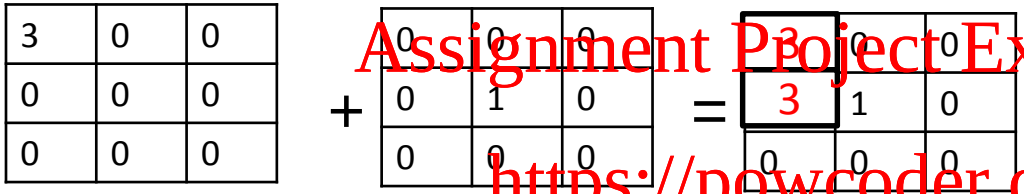








How to update the timetable?



- Max of all elements
- Max of times in local i'th row and remote k'th row.

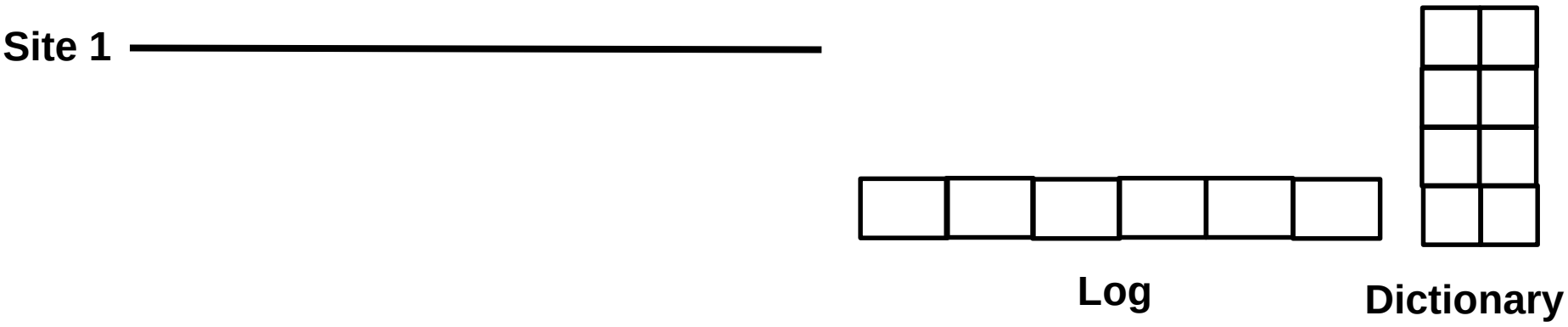
Log maintenance

- Initialize all entries in TT to 0.
- For each local operation, insert a copy in the local log.
- With each send operation from site i to site k: piggyback TT+ the following subset of the local log L_i: all records e such that hasrec(TT, e, k) is not true.
- On receipt of a message from site k by site i:
  - incorporate all new events into local log
  - update TT:
    - Max of all elements
    - Max of times in local i'th row and remote k'th row.

Dictionary problem

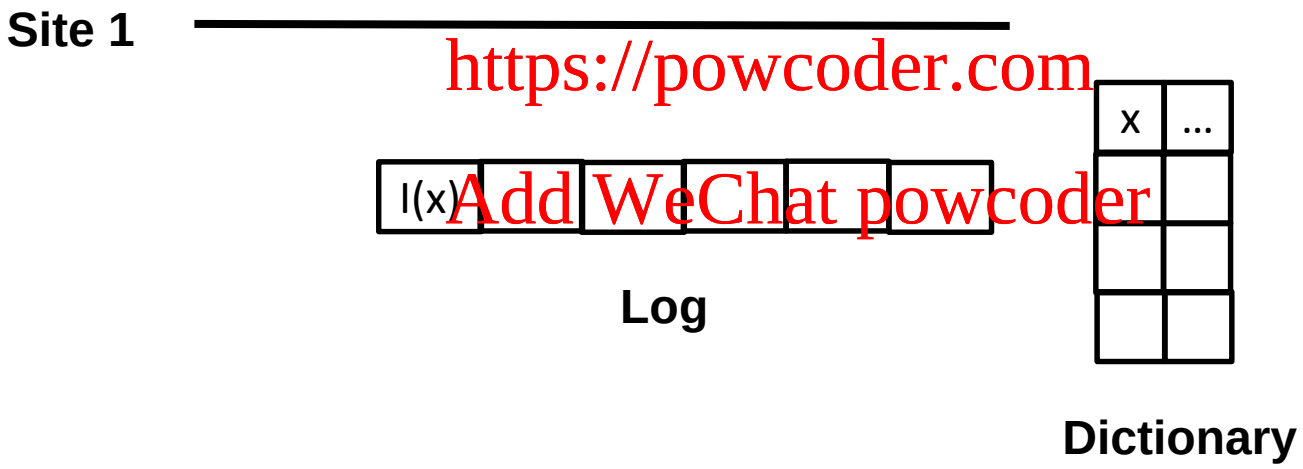
- Assume we want to maintain a *replicated dictionary* with *insert, delete and lookup*.
- On receipt of a message with partial log & TT:
  - Update local copy of the dictionary
  - Update local copy of TT as before
  - Garbage collect local log from any records that correspond to events e such that
    - **¬∃ site j such that hasrec(TT_i, e, j) is not true**

Replicated Dictionary- Example



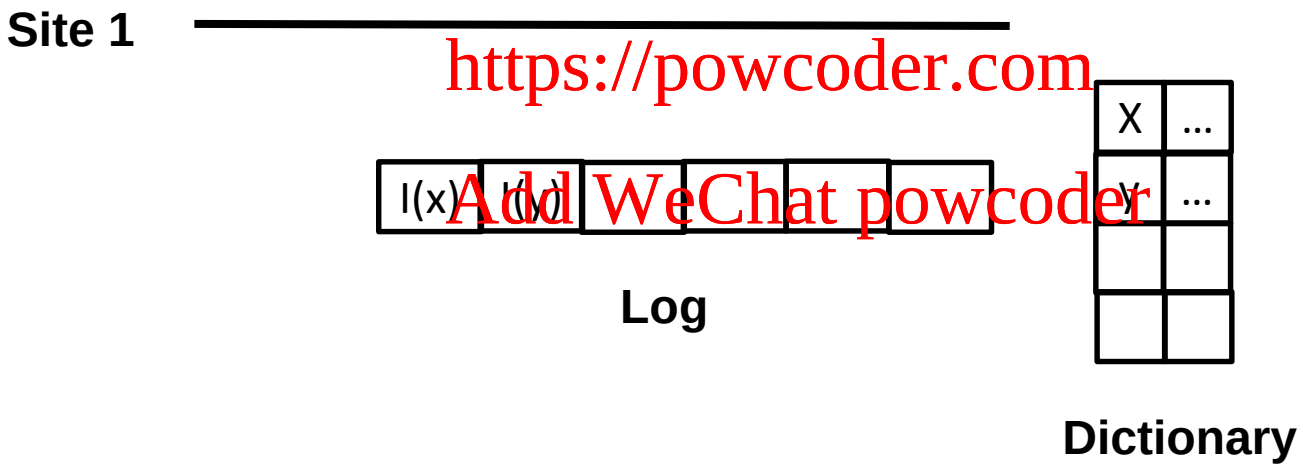
Replicated Dictionary- Example

- I(x)



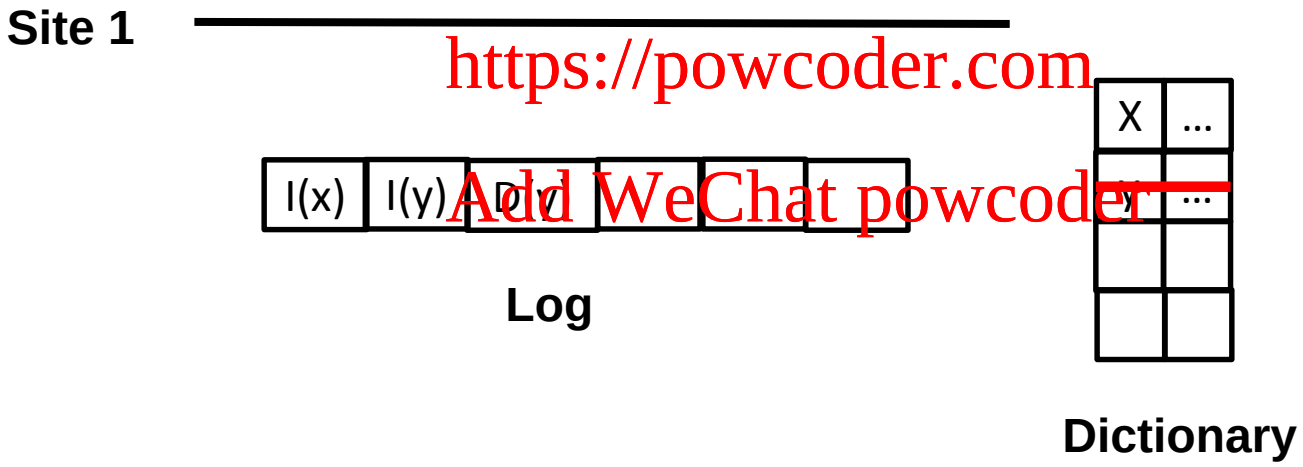
Replicated Dictionary- Example

- I(y)



Replicated Dictionary- Example

- D(y)



Replicated Dictionary- Example

- D(x)