ACMs – preliminaries

EEE8087

Dr Fei Xia and Dr Alex Bystrov

Prerequisites

- Petri nets
- State-transition diagrams
  - Reachability graphs
- Basic understanding of flip-flops, latches, registers and their operations
- Basic understanding of how clock signals are used in computing hardware

ACM resources

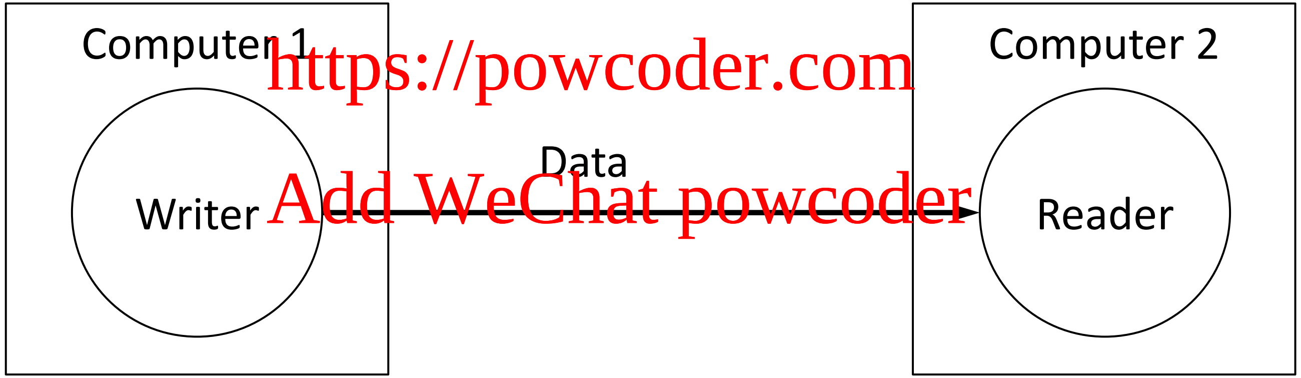
- Leslie Lamport on “atomic register” and Hugo Simpson on “asynchronous communication mechanisms”
  - Google, IEEE xplore, etc.
- Research papers on this subject by the µSystems Research Group, Newcastle University
  - Browse our publications in the Comfort and Coherent projects from http://async.org.uk/comfort/publications.html and http://async.org.uk/coherent/coherent_publications.html

Asynchronous Data Communications

- A complex system may include a number of different digital sub-systems, e.g. computers
- A major problem is how to transfer data from one sub-system to another
  - It boils down to data communications between two processes, each running on a different digital computation device, e.g. computer
  - We view these processes as a 'writer' and a 'reader' and the goal is to transfer data from the writer to the reader correctly and efficiently

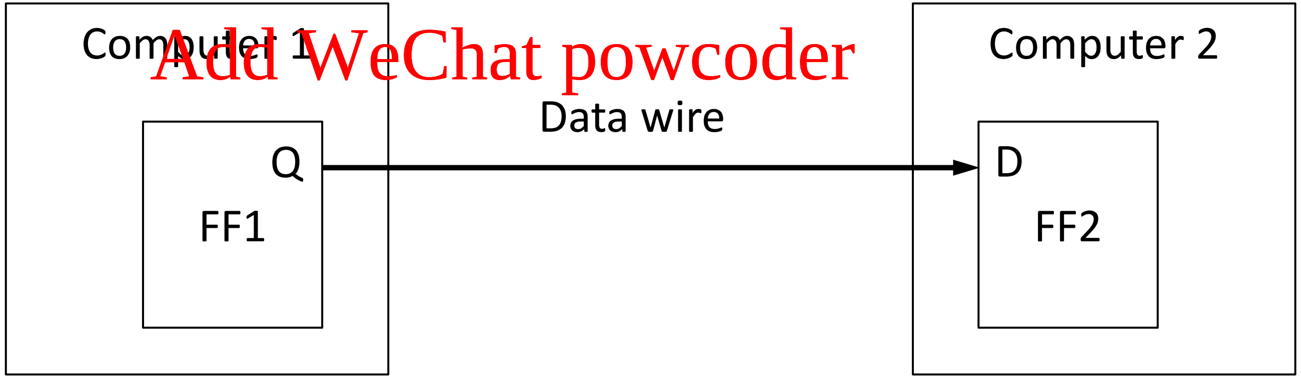
Asynchronous Data Communications

- Schematic diagram of the data transfer



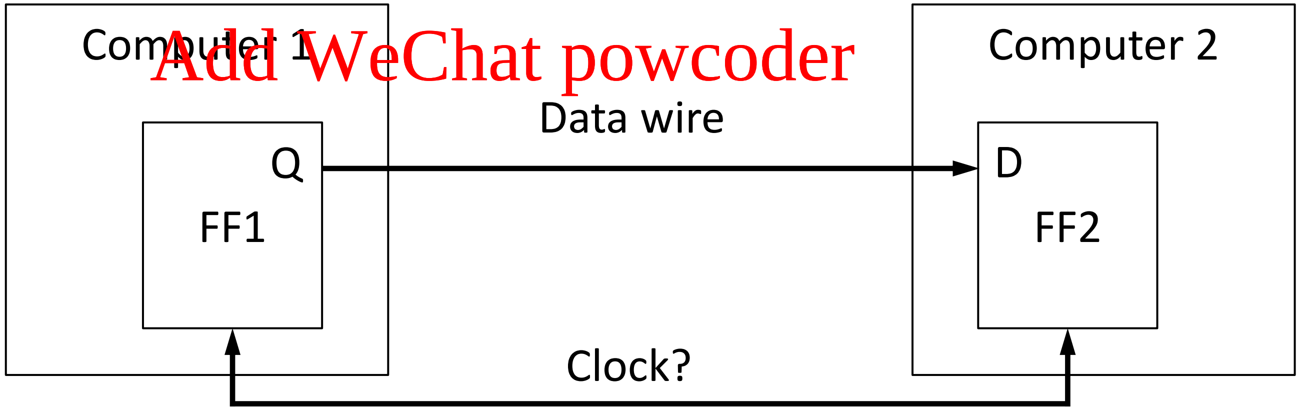
Fundamental Problem

- Synchronization
  - Different computers tend to be run on different clocks
  - But at the basic level, a bit of data is sent/received through the smallest memory, i.e. flip-flops or latches



Fundamental Problem

- Synchronization
  - The latches involved need to be on the same clock
  - This 'same clock' needs to be the clock of Computer 1 or Computer 2 but these two are not the same!



Synchronization for data transfer

- Synchronizers
  - Devices that make the two sides of the data transfer (writer and reader) run on *effectively the same clock*
  - Waiting
  - Losing data correctness
- Problems
  - Metastability means that for 100% correctness you need waiting of unbounded time
  - Waiting only for bounded time you have non-zero probabilities of data errors

Buffering

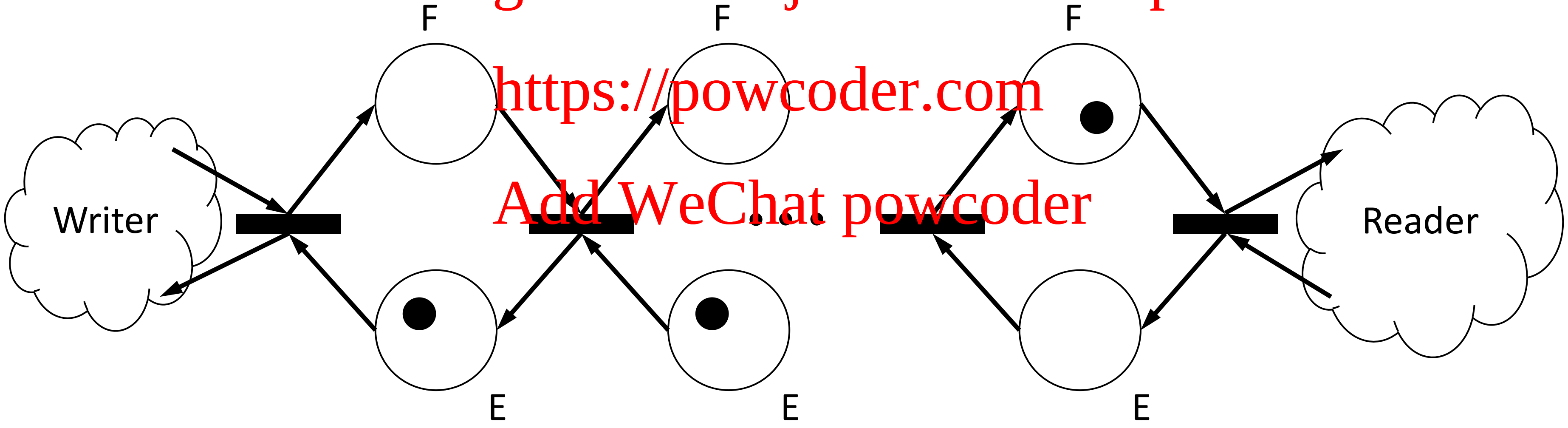
- Waiting, if unbounded, violates real-time requirements
- Insert a buffer between reader and writer so either side can move on without waiting for the other side?
- Let’s look at the Petri net model of an *n*-space buffer for asynchronous data communication

Typical *n*-space buffers

- First in first out (FIFO)
  - This is the most common buffer between two different digital devices
- Last in first out (LIFO)
  - Also known as a stack, this is less common between two different digital devices
- Random access memory (RAM)
  - Generally known as a bag, if access is truly random there is very little real use for it

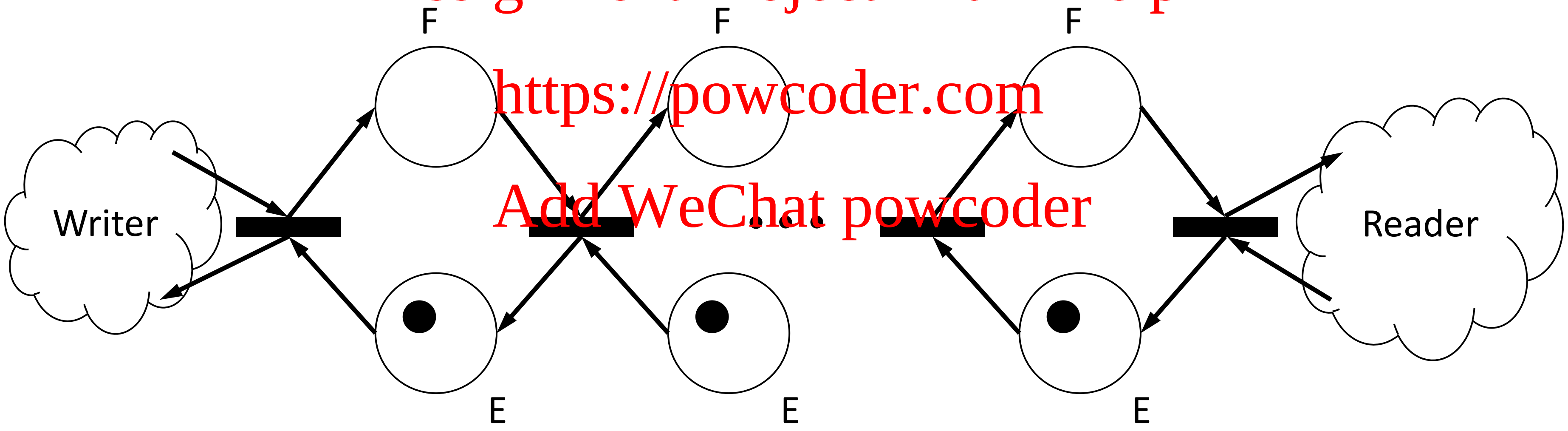
Petri net model of a FIFO buffer

- FIFO with *n* spaces between writer and reader



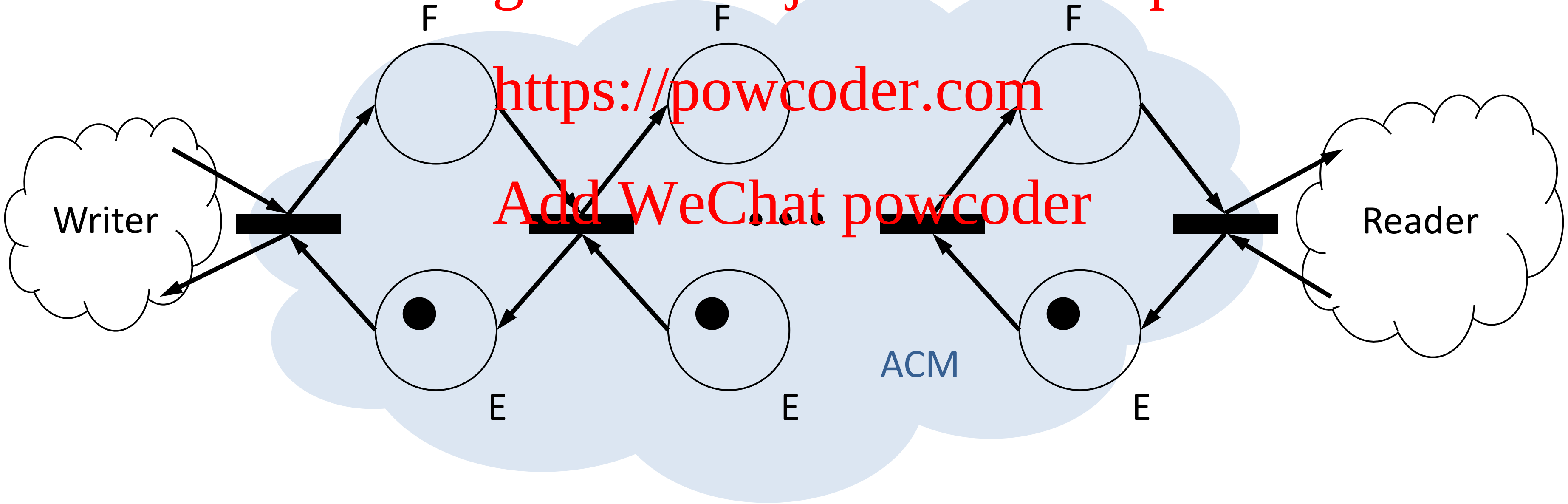
Each space may be empty (E) or full (F)

- The entire buffer may be empty (reader waits) or full (writer waits)



ACM

- Asynchronous communication mechanism between reader and writer



ACM

- For fully asynchronous data communication
- No waiting by either writer or reader, single-space buffer
- Wide usage in real-time systems
  - Example: public clocks
  - 'Writer' is the clock's internal mechanism
  - 'Readers' are people looking at the clock
  - If nobody looks at the clock during an update cycle, it updates anyway – the 'lost' value is not used
  - If people look at the clock during an update cycle, the 'data' stays the same until the next update
  - If someone reads the clock multiple times during an update cycle, they read the same value – the same value is re-used

09:58

Full reader/writer asynchrony

- Known by a number of different names, but the same thing conceptually
  - Leslie Lamport: Atomic register https://lamport.azurewebsites.net/pubs/interprocess.ps
  - Hugo Simpson: Pool (four-slot ACM) https://ieeexplore.ieee.org/document/41349
- Writer updates value (overwriting)
- Reader does not modify value (re-reading)
- Buffer always contains one valid data item
  - Initialized with valid data value before a run
- Synchronized to the writer when writing, to the reader when reading
  - But how to accommodate simultaneous reading/writing?

ACM preliminaries recap

- No ACM
  - Fully synchronized data transfer – reader and writer must be on the same clock during data transfer and reading and writing happens on the same data at the same time
- Traditional buffers (FIFO etc.)
  - Some asynchrony allowed for data transfer – reader and writer usually do not access the same data item at the same time
  - Waiting cannot be fully avoided
- Fully asynchronous ACM
  - No waiting by either side, data always valid