COS3043
System Fundamentals

Lecture 6

Topics

1.	Abstractions 1.1 Hardware Resources 1.2 OS Functionality 1.3 Managing the CPU and Memory
2.	OS Structure 2.1 SPIN Approach 2.2 Exokernel Approach 2.3 L3/L4 Micro-Kernel Approach
3.	Virtualization 3.1 Intro to Virtualization 3.2 Memory Virtualization 3.3 CPU and Device Virtualization
4.	Parallelism 4.1 Shared Memory Machines 4.2 Synchronization 4.3 Communication 4.4 Scheduling
5.	Distributed Systems 5.1 Definitions 5.2 Lamport Clocks 5.3 Latency Limit

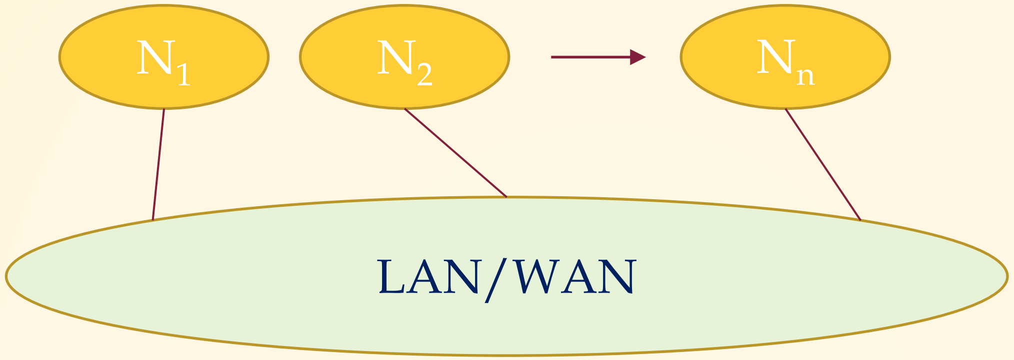
6.	Distributed Object Technology 6.1 Spring Operating System 6.2 Java RMI 6.3 Enterprise Java Beans
7.	Design and Implementation of Distributed Services 7.1 Global Memory System 7.2 Distributed Shared Memory 7.3 Distributed File System
8.	System Recovery 8.1 Lightweight Recoverable Virtual Memory 8.2 Rio Vista 8.3 Quicksilver
9.	Internet Scale Computing 9.1 GiantScale Services 9.2 Content Delivery Networks 9.3 MapReduce
10.	Real-Time and Multimedia 10.1 Persistent Temporal Streams

List of Discussion

- Distributed System Basics
- Lamport Clocks
- Latency Limits

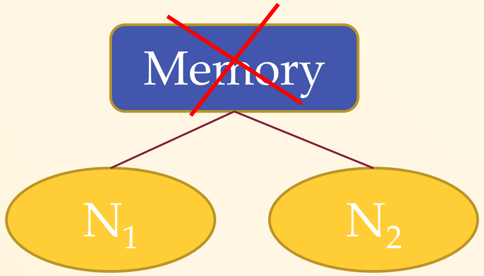
Distributed System Basics

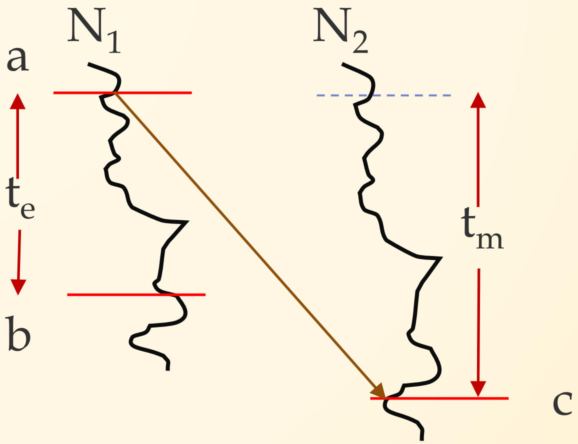
Distributed Systems Definition



Fibre, cable, satellite…

- No physical shared memory between nodes.
- Communication via message between nodes.

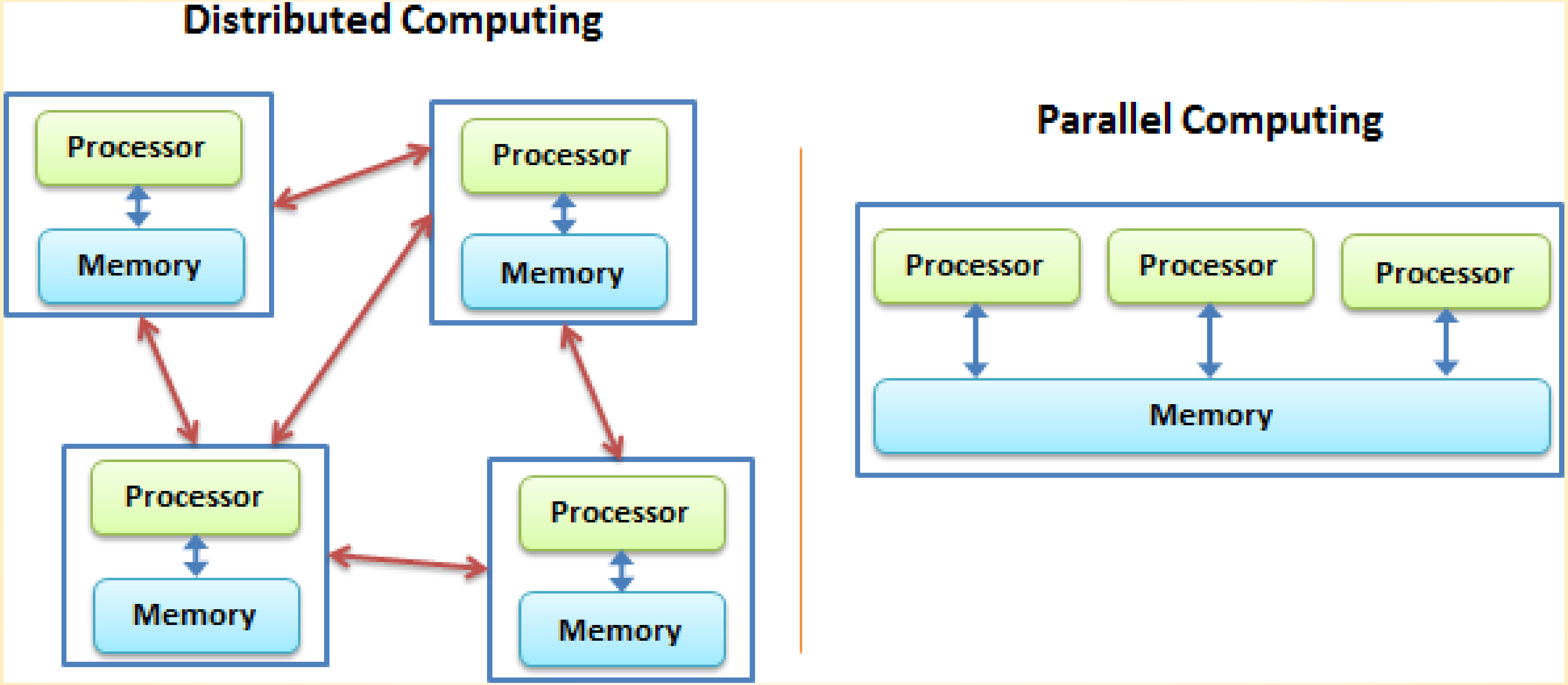




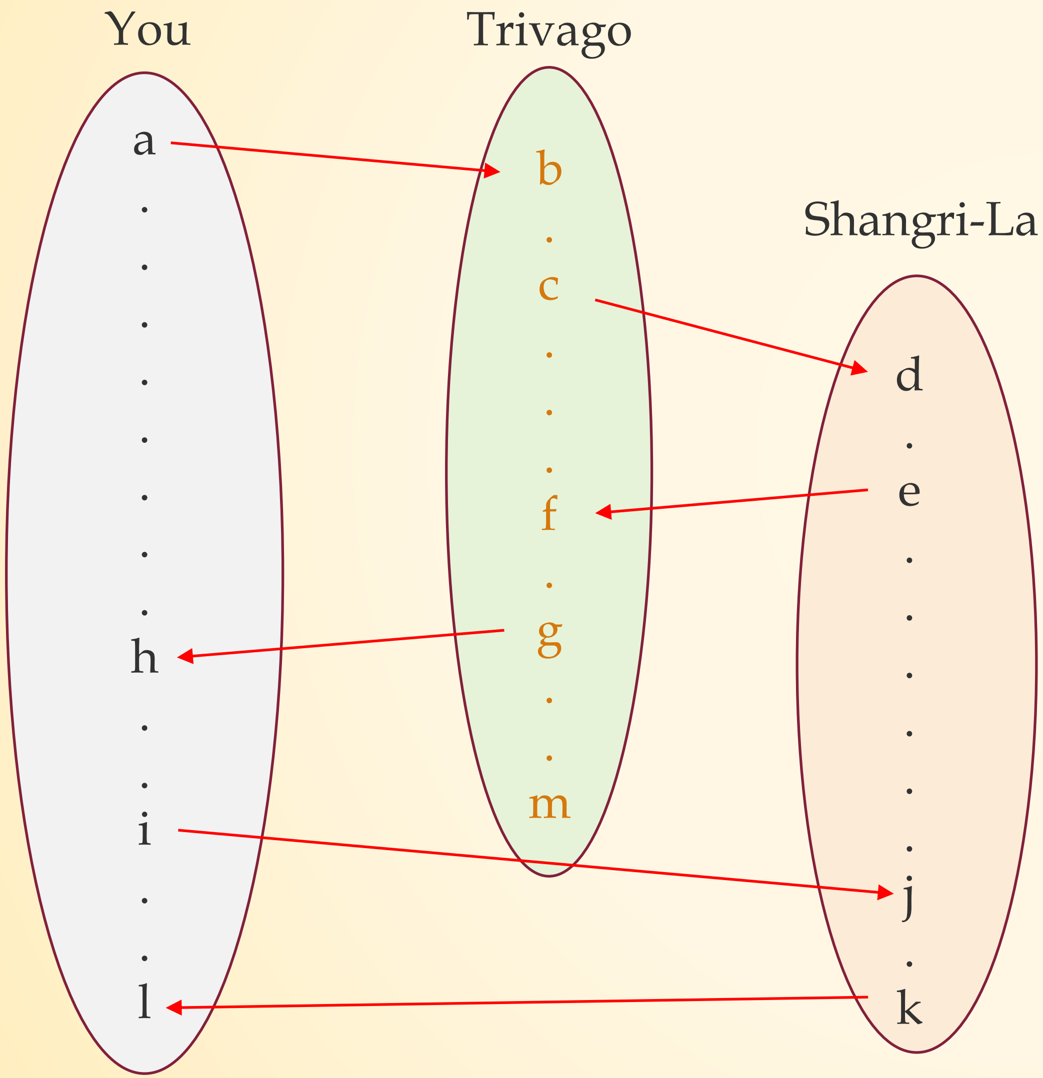
$t_m^{a-c} >> t_e^{a-b}$

- Message communication time is significantly larger than event computation time

Distributed Systems Versus Parallel Systems



Nodes Communication Example



Beliefs:

- Processes are sequential
  - Events totally ordered
  - h → i, f → g, d → e…
- Send before receive
  - a → b, c → d, e → f…

"Happened Before Relationship"

We use a → b to show a happened before b.

Happened Before Relationship

$a \rightarrow b$ => either a & b are same process or communication where b happened after a.

Transitivity of "happened before"

$a \rightarrow b$

$b \rightarrow c$

=> $a \rightarrow c$

Question

Consider the following set of events:

N1		N2
f	$\longrightarrow$	b
a		g

What can you say about relationship between ‘a’ & ‘b’:

- a $\rightarrow$ b
- b $\rightarrow$ a
- Either
- Neither

Happened Before Relationship

$a \rightarrow b$ => either a & b are same process or communication where b happened after a.

Transitivity of "happened before"

$a \rightarrow b$

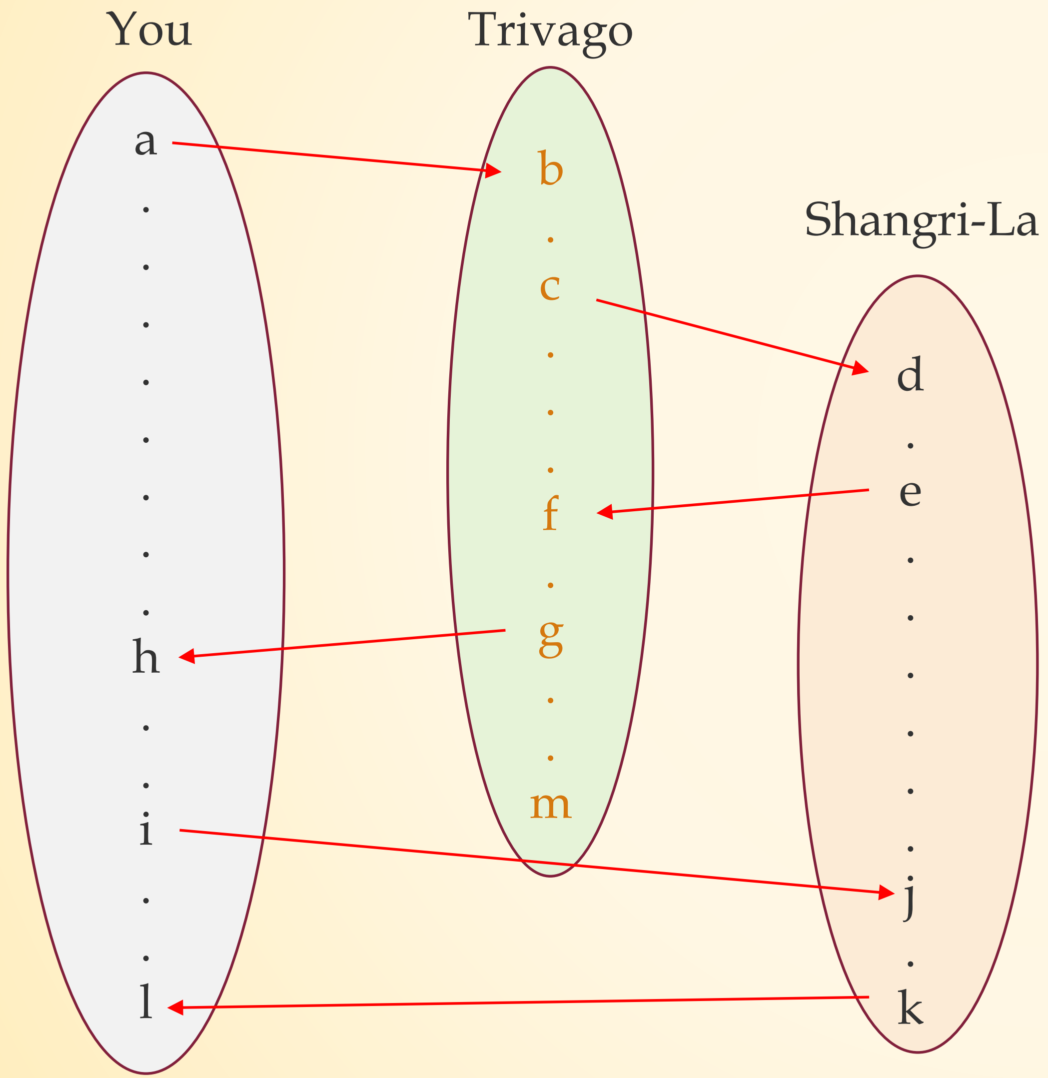
$b \rightarrow c$

=> $a \rightarrow c$

Concurrent events (when a & b have no relationship)

$a \,||\, b$

Exercise



Identify the events by $\rightarrow$ and $||$

$a \rightarrow h \rightarrow i \rightarrow l$
$b \rightarrow c \rightarrow f \rightarrow g \rightarrow m$
$d \rightarrow e \rightarrow j \rightarrow k$

$a \rightarrow b$
$c \rightarrow d$
$e \rightarrow f$
$g \rightarrow h$
$i \rightarrow j$
$k \rightarrow l$

$h \; || \; m$
$i \; || \; m$
$l \; || \; m$
$m \; || \; j$
$m \; || \; k$

How about transitive connected relationship?

Lamport Clocks

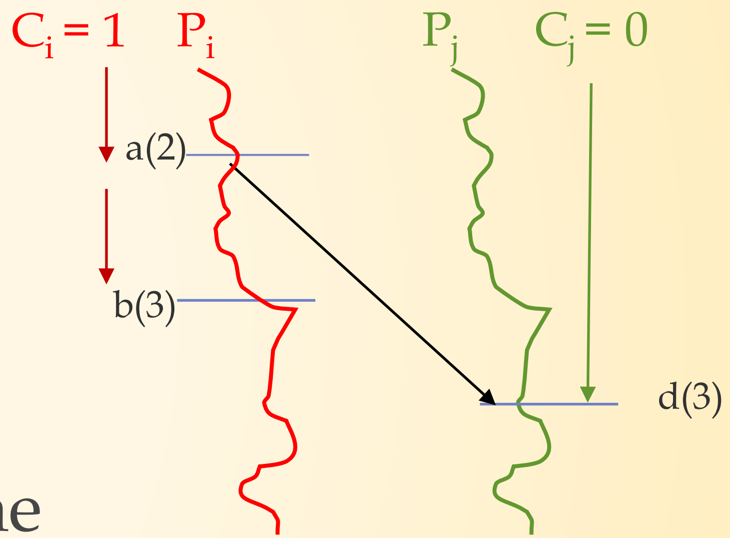
Lamport's Logical Clock

- Each node:
  - ➢Knows its own events
  - ➢Knows its communication events (in/out)
- Lamport's logical clock:
  - ➢Monotonic increase of own event times

    Condition 1: $C_i(a) < C_i(b)$
  - ➢Message receipt time greater than sent time

    Condition 2: $Ci(a) < Cj(d)$

    => Choose: $C_j(d) = Max(C_i(a){+}{+}, C_j)$
  - ➢Timestamp of concurrent events?



Question

$C(a) < C(b)$

Based on the above info, what can we conclude?

- $a \rightarrow b$
- $b \rightarrow a$
- Either
- Neither
- $a \rightarrow b$ if (…)

Question (Answer)

$C(a) < C(b)$

Based on the above info, we can conclude that:

- a $\rightarrow$ b , if and only if:

  a & b events are in the same process

  or

  a is a send event & b is the corresponding receipt event

Logical Clock Conditions

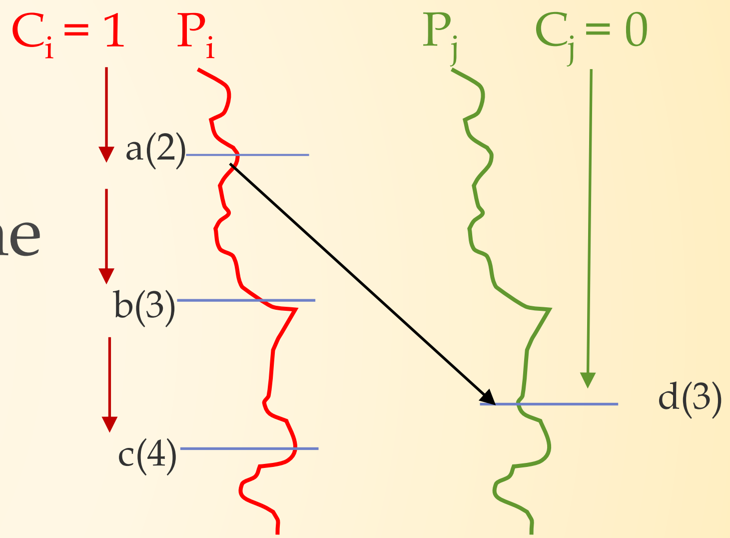
- Lamport's logical clock:
  - Monotonic increase of own event times

    Condition 1: $C_i(a) < C_i(b)$
  - Message receipt time greater than sent time

    Condition 2: $Ci(a) < Cj(d)$

    => Choose: $C_j(d) = Max(C_i(a)++, C_j)$
  - Timestamp of concurrent events (b & d)

    Arbitrary timestamp
  - $C(x) < C(y)$!= $x \rightarrow y$



Need for Total Order

- So far, based on what we have gone through such as the "happened before" and Lamport, it seems that it's good enough for distributed system to handle the communications.
- However in actual fact, it's imperative to have a "total order" because in some situations.
- Real life example: Family Car Usage for 4 persons
  - Test everyone when one of them wants to use.
  - The earliest timestamp wins.
  - But how about if same timestamp?

    => Age wins (when there is a tie).

Lamport's Total Order

- Condition for =>

    a => b if

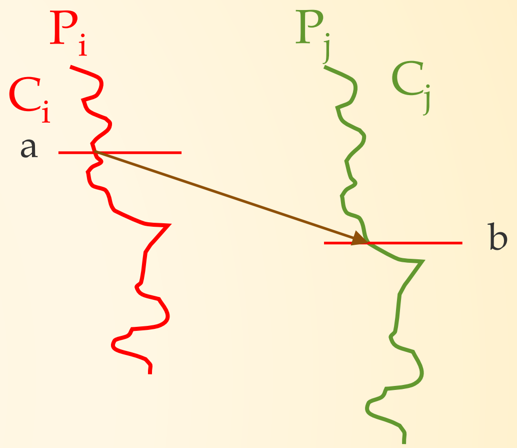
    $C_i(a) < C_j(b)$

    OR

    $C_i(a) = C_j(b)$ and $P_i << P_j$

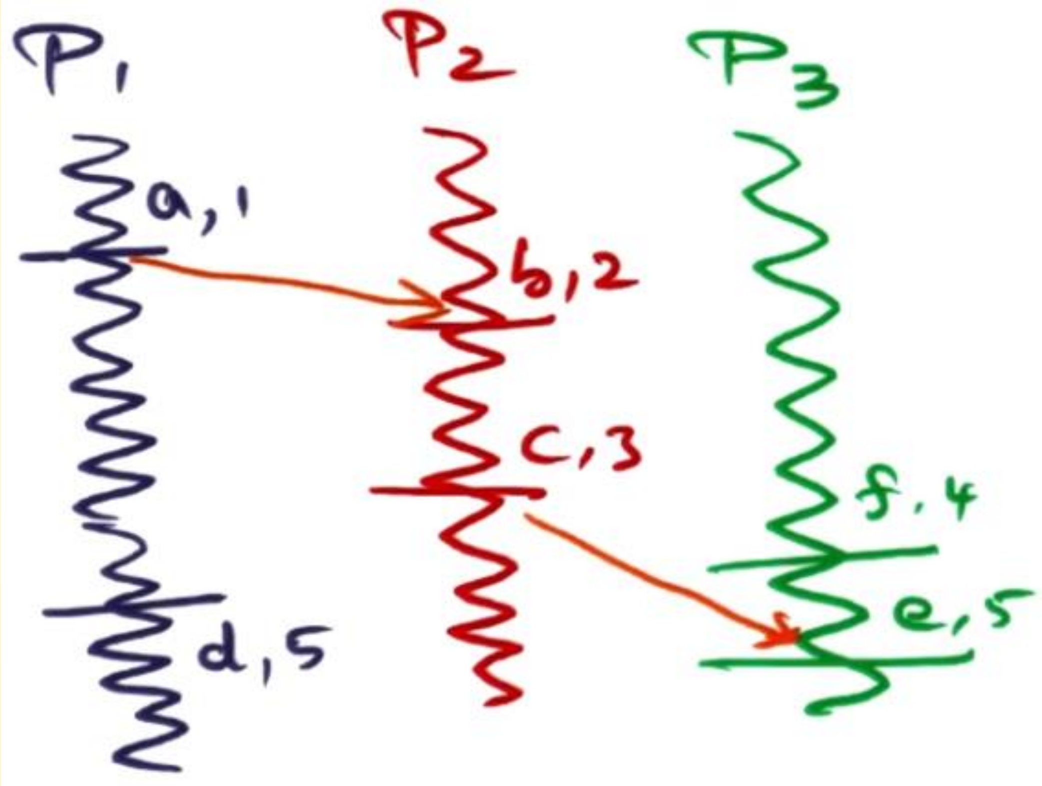
    C <<< arbitrary "well known" condition to break a tie

    Example: process ID to determine



- No single total order.

Question?

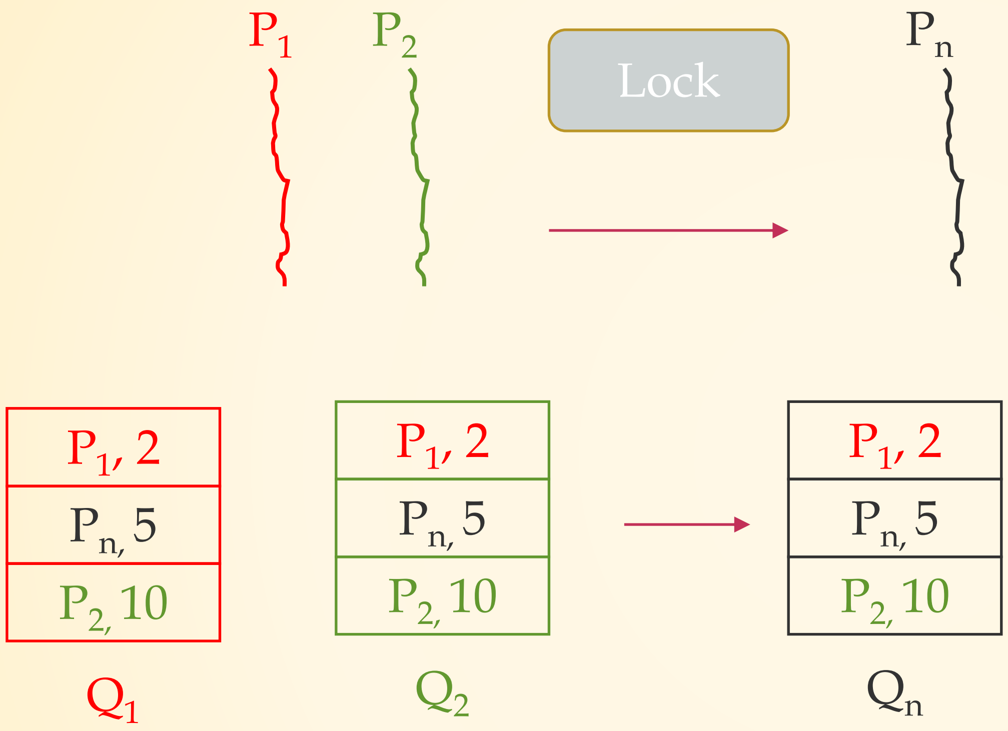


Referring to the diagram, what is the total order using process ID to break the tie?

Answer:
Total order respecting timestamps and process ID

a => b => c => f => d => e

Distributed Mutual Exclusive Lock Algorithm



Distributed Mutual Exclusive Lock Algorithm

- Correctness?
  - Messages arrive in order
  - No message loss
  - Q's totally ordered => by Lamport's logical clock + PID to break tie.

Question

How many messages exchanged among all the nodes (N), for each lock acquisition and followed by the release of the lock?

- N
- N-1
- 2 (N-1)
- 3 (N-1)
- 4 (N-1)

Message Complexity

Lock(L) => N-1 Request Message

=> N-1 Acknowledge Message

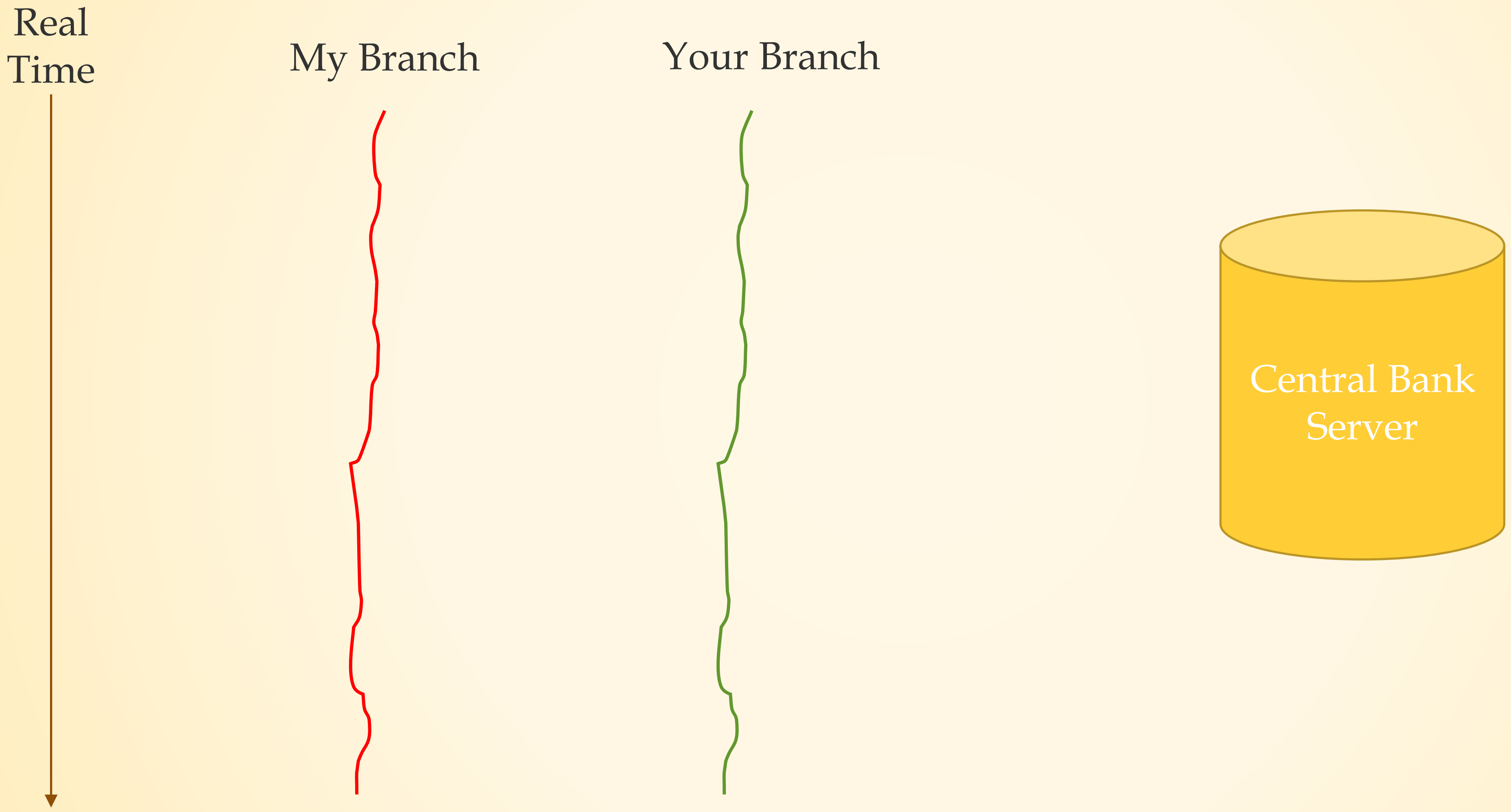
Unlock(L) => N-1 Unlock Message

TOTAL = 3(N-1) messages

Can we improve the timing?

- Yes, defer Ack message if my Request time precedes yours. After I am done with the critical section, only send Ack message with the unlock step => now reduced to 2(N-1) messages only.
- Ricart-Agrawala lock algorithm – Please read about this algorithm on your own.

Real World Scenario



Lamport's Physical Clock

a $\mapsto$ b

=> $C_i(a) < C_j(b)$

Physical clock conditions:

- PC1 - bound on individual clock drift

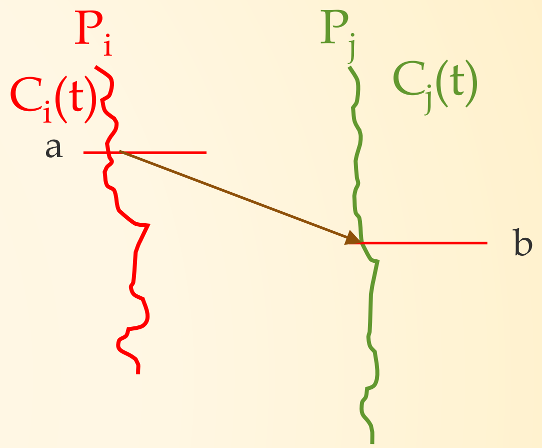
  The clock drift C_i is very small compared to real clock.

  $$\frac{dC_i(t)}{dt} - 1 < k \qquad \text{and} \qquad k << 1$$

- PC2 - bound on mutual clock drift

  The clock difference between all P's (i.e. the mutual clock drift) is very small.

  $C_i(t) - C_j(t) < \varepsilon$

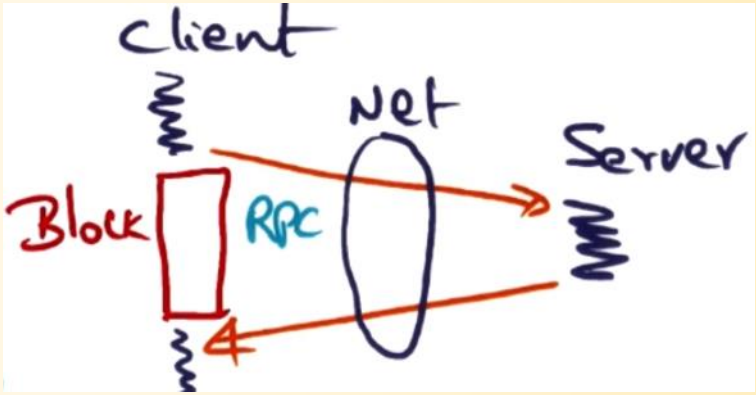


Both k and ε bound values must always be negligible compared to IPC time so that there is no anomaly in the system as seen in our discussion.

Latency Limits

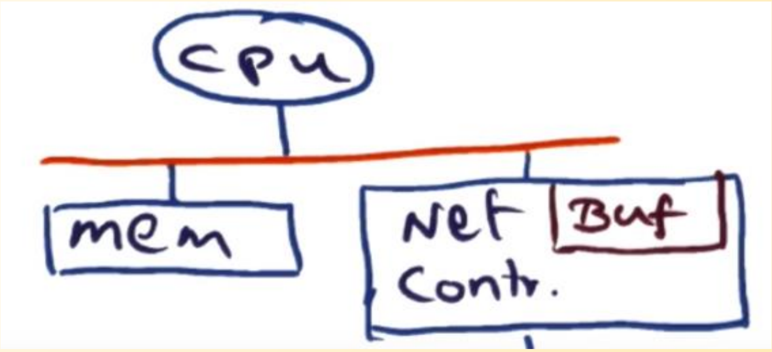
Latency Versus Throughput

- Latency => elapsed time
- Throughput => events per unit time (bandwidth is throughput measure)



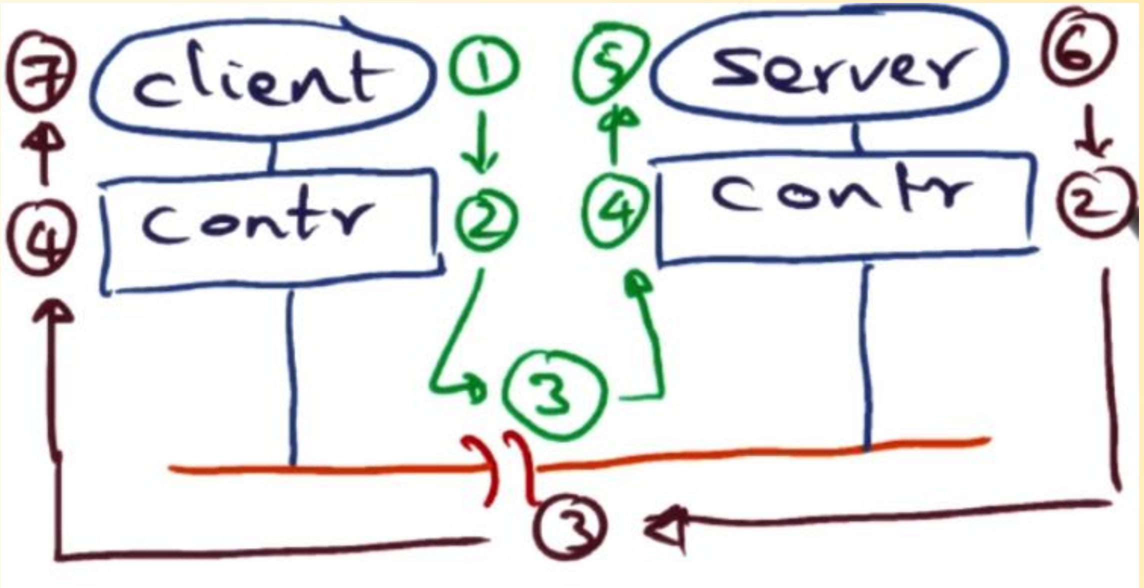
Example for discussion:

- Remote Procedure Call (RPC) Performance
  - ➢Hardware overhead
  - ➢Software overhead
- Focus on this lesson: how to reduce software overhead.



Components of RPC Latency

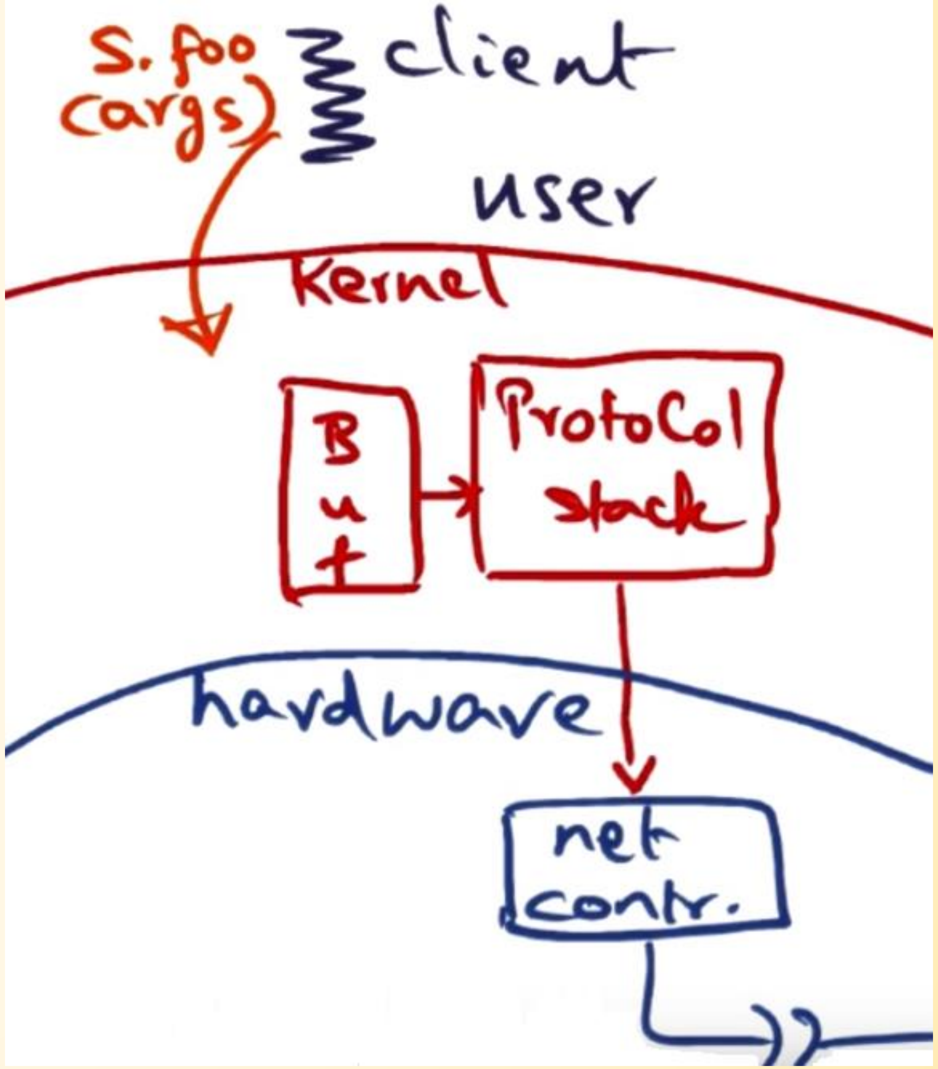
1. Client call
2. Controller latency
3. Time on wire
4. Interrupt handling
5. Server setup to execute call
6. Server execution + reply
7. Client setup to receive results and restart



Sources of Overheard in RPC

- Marshalling
- Data Copying
- Control Transferring
- Protocol Processing

- How to reduce kernel overhead?
  - Take what hardware can offer to reduce latency (as not much more can be done to reduce the hardware latency)
  - Will focus on software overhead.

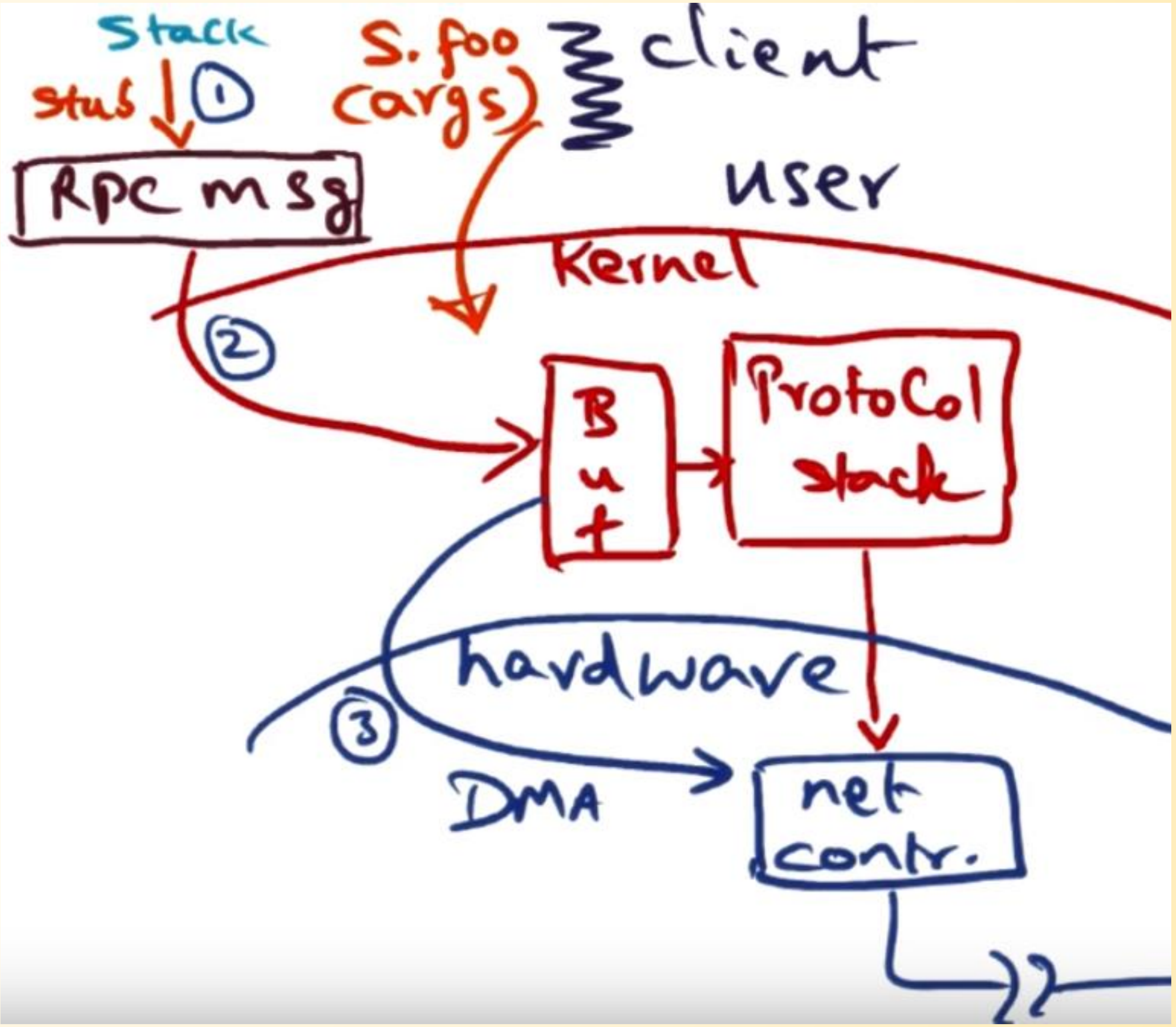


Marshalling & Data Copying

- Three copies:
  - ➢Client stub
  - ➢Kernel buffer
  - ➢DMA controller

- How to reduce?
  - ➢Marshal in the kernel buffer directly

    Or
  - ➢Shared descriptors between client stub and kernel



Control Transferring

- 4 contacts of control transfer:

- Cutdown contacts to 2:

Protocol Processing

- What 'transport' for RPC?
  - ➢LAN reliable => reduce latency.
- Choices of reducing latency in transport:
  - ➢No low level ACKS
  - ➢Hardware checksum for packet integrity
  - ➢No client side buffering since client is blocked
  - ➢Overlap server side buffering with result transmission.