# Time in Distributed Systems

* There is no common universal time (Einstein)

  the speed of light is constant for all observers
  irrespective of their velocity

* Assume our DS is earth-based

* Earth-time is defined w.r.t. the earth's rotation
  - solar year is constant
  - solar day is lengthening (earth slowing)

  from 1948, earth time has been based on
  atomically-defined caesium clocks
  (atomic second = solar second)

  there are now about 50 such clocks
  average value = TIA (International Atomic Time)
  BIH (Int'nl Bureau d'Heure announces
  leap seconds to keep in phase with the sun
  - about 30 so far, last in Jan 99

* UTC (Universal Coordinated Time) is corrected TIA

* UTC services are offered by radio stations
  and satellites
  - receivers are available commercially
  - accuracy varies with weather conditions
    stated bounds are 1ms - 10ms
    radio: Rugby, Fort Collins Colorado 10ms
    satellite: GOES 0.5ms, GPS  1ms

* UTC signals take time to propagate
  - we can define a time interval
    UTC-d1 -> UTC+d2
    during which an event has happened
    w.r.t. earth time



# Timers in computers

* based on frequency of oscillation of
  a quartz crystal

* Each computer has a timer which interrupts
  periodically. In practice, the number of interrupts
  per hour varies slightly, and with temperature,
  and clocks may drift.
  typically 1/10*6  1 sec in 11.6 days

* note that computer systems tag events
  with timestamps

---

How is time used in distributed systems?

Do we need accurate time?

What does "A happened before B" mean
    in a distributed system?

We have seen that we cannot know the time at
which an event occurs accurately, but only
specify an interval. We now have to increase
that interval to allow for clock drift and
any other sources of inaccuracy.

If two events have timestamps which differ by
less than some interval value we CAN'T SAY
in which order they occurred.

## Use of time in distributed systems: Examples

1. Any source of resource contention
   e.g. airline booking

   if the reservation requests for two transactions may each be satisfied but there are not enough seats left for both, then the transaction with the earliest timestamp wins.

   - we don't need fine-grained accuracy
     we just need a timestamp ordering convention
     so we agree who won on a tie

2. programming environments e.g. UNIX make

   suppose a make (compile and link) involves many components which are edited on distributed computers

   suppose a component is edited immediately after a make but on a computer with a slow clock and the edited source is given a timestamp earlier than the make

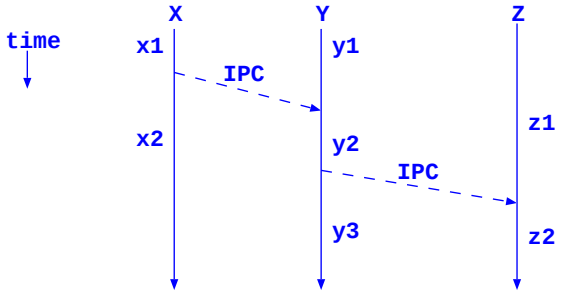
   on the next make, this component is not recompiled

   - this may seem unlikely to happen,
     provided we ensure clocks are initialised
     reasonably accurately
     e,g, not from the operator's watch

   - note that we are seeing an example of the importance of event ordering: did the edit take place before or after the last make?



## Event ordering in distributed systems




define < to mean "happened before"

suppose inter-process communication (IPC) takes place

* events within a single process are ordered

* events in region x1 < events in regions y2, y3

* events in region x1 < events in regions z2

* events in region y1, y2 < events in regions z2

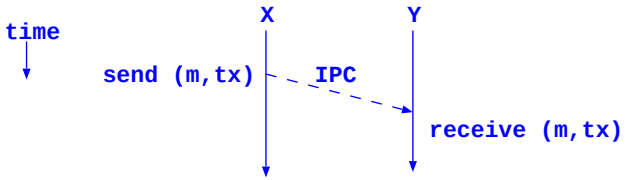
* for events in other regions we CAN'T SAY
  e.g. events in x1 and y1, y1 and z1 etc.

IPC defines a PARTIAL ORDERING
on the events in the DS

note that this ordering is true whatever the local clocks of X, Y and Z indicate

we must ensure that the values of the local clocks
respect this event ordering

suppose a message m is timestamped tx by X on sending




note that X's send CAUSED Y's receive

suppose Y's local clock reads ty on receive(m,tx)

if ty > tx OK

if ty < = tx reset ty to tx + one increment

This imposes logical time on the system
(sometimes called Lamport time due to Lamport's
classic paper on this topic)

BUT - system time adjusted in this way will drift
ahead of UTC

we could use counters rather than timestamps if all
we need is event ordering

How can we generate timestamps that are:
- reasonably close to UTC
- preserve causal ordering



# Protocols for synchronising physical clocks

## Cristian's algorithm 1989

* assume one machine has a UCT receiver (time server)

* each machine polls the time server periodically
(period depends on max. clock drift and
accuracy required)

* server sends back its value of the time

* client receives this value and may:

- use it as it is
- add the known minimum network delay
- add half the time between this send and receive

* now consider resetting the receiver's local clock

if value >= local-time OK
use it to set the clock or adjust the interrupt rate
for a while to speed up the clock (e.g. 10ms - 9ms)

NOTE that time can't be put back or event ordering
within the local system would become incorrect

if value < local time
adjust the interrupt rate to slow down the clock
(e.g. 10ms-11ms)

* a number of time servers can be used
for reliability

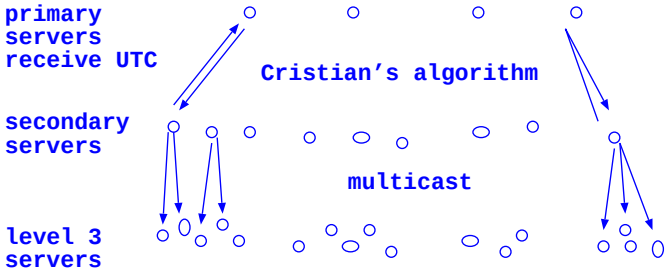
CONSIDER the accuracy of any timestamp -
the length of the interval
UTC-d1, UTC+d2
during which an event occurs

## Berkeley UNIX (Gusella & Zatti, 1989)

* if no machines have receivers...
* a nominated "time-server" asks all machines for their times
* computes the average value
* broadcasts to all machines
* operator may set the time manually from time to time

## NTP (Network Time Protocol) (Mills, 1991)

* for the Internet as a hierarchy of computers




* uses UDP
* allow for estimated network delay and adjust clocks as described above
* accurate to a few tens of ms



# Time in Distributed Systems

* There is no common universal time (Einstein)

  the speed of light is constant for all observers irrespective of their velocity

* Assume our DS is earth-based

* Earth-time is defined w.r.t. the earth's rotation
  - solar year is constant
  - solar day is lengthening (earth slowing)

  from 1948, earth time has been based on atomically-defined caesium clocks (atomic second = solar second)

  there are now about 50 such clocks
  average value = TIA (International Atomic Time)
  BIH (Int'nl Bureau d'Heure announces leap seconds to keep in phase with the sun
  - about 30 so far, last in Jan 99

* UTC (Universal Coordinated Time) is corrected TIA

* UTC services are offered by radio stations and satellites
  - receivers are available commercially
  - accuracy varies with weather conditions
    stated bounds are 1ms - 10ms
    radio: Rugby, Fort Collins Colorado 10ms
    satellite: GOES 0.5ms, GPS 1ms

* UTC signals take time to propagate
  - we can define a time interval
    UTC-d1 -> UTC+d2
    during which an event has happened
    w.r.t. earth time

## Timers in computers

* based on frequency of oscillation of a quartz crystal

* Each computer has a timer which interrupts periodically. In practice, the number of interrupts per hour varies slightly, and with temperature, and clocks may drift.
typically 1/10*6  1 sec in 11.6 days

* note that computer systems tag events with timestamps

---

How is time used in distributed systems?

Do we need accurate time?

What does "A happened before B" mean in a distributed system?

We have seen that we cannot know the time at which an event occurs accurately, but only specify an interval. We now have to increase that interval to allow for clock drift and any other sources of inaccuracy.

If two events have timestamps which differ by less than some interval value we CAN'T SAY in which order they occurred.



## Use of time in distributed systems: Examples

1. Any source of resource contention
   e.g. airline booking

   if the reservation requests for two transactions may each be satisfied but there are not enough seats left for both, then the transaction with the earliest timestamp wins.

   - we don't need fine-grained accuracy
     we just need a timestamp ordering convention
     so we agree who won on a tie

2. programming environments e.g. UNIX make

   suppose a make (compile and link) involves many components which are edited on distributed computers

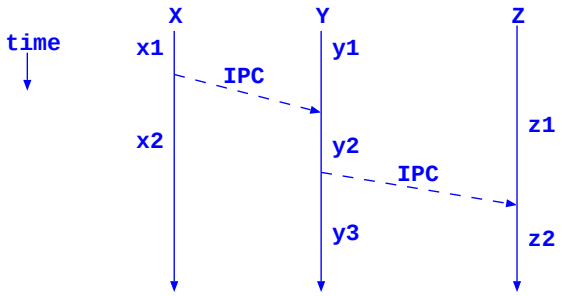
   suppose a component is edited immediately after a make but on a computer with a slow clock and the edited source is given a timestamp earlier than the make

   on the next make, this component is not recompiled

   - this may seem unlikely to happen,
     provided we ensure clocks are initialised reasonably accurately
     e,g, not from the operator's watch

   - note that we are seeing an example of the importance of event ordering: did the edit take place before or after the last make?

## Event ordering in distributed systems




define < to mean "happened before"

suppose inter-process communication (IPC) takes place

* events within a single process are ordered

* events in region x1 < events in regions y2, y3

* events in region x1 < events in regions z2

* events in region y1, y2 < events in regions z2

* for events in other regions we CAN'T SAY
  e.g. events in x1 and y1, y1 and z1 etc.

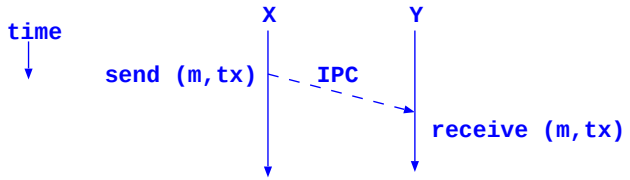
IPC defines a PARTIAL ORDERING
on the events in the DS

note that this ordering is true whatever the
local clocks of X, Y and Z indicate



we must ensure that the values of the local clocks
respect this event ordering

suppose a message m is timestamped tx by X on sending




note that X's send CAUSED Y's receive

suppose Y's local clock reads ty on receive(m,tx)

if ty > tx OK

if ty < = tx reset ty to tx + one increment

This imposes logical time on the system
(sometimes called Lamport time due to Lamport's
classic paper on this topic)

BUT - system time adjusted in this way will drift
ahead of UTC

we could use counters rather than timestamps if all
we need is event ordering

How can we generate timestamps that are:
- reasonably close to UTC
- preserve causal ordering

# Protocols for synchronising physical clocks

## Cristian's algorithm 1989

* assume one machine has a UCT receiver (time server)

* each machine polls the time server periodically (period depends on max. clock drift and accuracy required)

* server sends back its value of the time

* client receives this value and may:

  - use it as it is
  - add the known minimum network delay
  - add half the time between this send and receive

* now consider resetting the receiver's local clock

  if value >= local-time  OK
  use it to set the clock or adjust the interrupt rate
  for a while to speed up the clock (e.g. 10ms - 9ms)

NOTE that time can't be put back or event ordering within the local system would become incorrect

  if value < local time
  adjust the interrupt rate to slow down the clock
  (e.g. 10ms-11ms)

* a number of time servers can be used for reliability

CONSIDER the accuracy of any timestamp -
the length of the interval
UTC-d1, UTC+d2
during which an event occurs



## Berkeley UNIX (Gusella & Zatti, 1989)

* if no machines have receivers...

* a nominated "time-server" asks all machines for their times

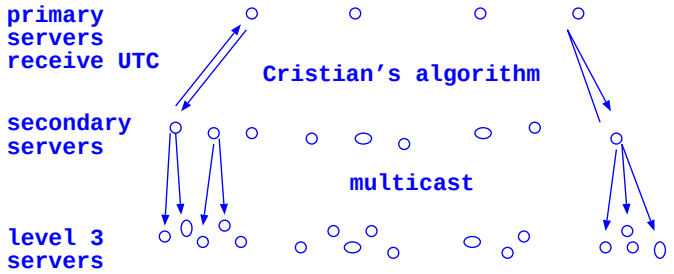
* computes the average value

* broadcasts to all machines

* operator may set the time manually from time to time

## NTP (Network Time Protocol) (Mills, 1991)

* for the Internet as a hierarchy of computers




* uses UDP

* allow for estimated network delay and adjust clocks as described above

* accurate to a few tens of ms