

Time and Clocks

Some of this material is based on [?, Kopetz, Ch. 3].
NOTE: *These notes are not complete. More to come.*

Global Time and Events

Global time is the exact time reference of an omniscient external observer, represented by a real half-line $\text{TIME} = [0, \infty)$. The the origin 0 is fixed at a standard point in the remote past, before any moment of interest occurs. The unit of measure for TIME is the second (s), but we are often interested in fractional units, such as *milliseconds* ($ms = 10^{-3}s$), *microseconds* ($\mu s = 10^{-6}s$), *nanoseconds* ($ns = 10^{-9}s$).

Conceptually, an *event* e is an observable condition to which we can associate an instant in TIME . Events have no duration, or more precisely, their *observation* is instantaneous. Later on (but not in this chapter) the notion of an event is generalized to include *compound events* whose conditions include prior events. However, even compound events are associated with instantaneous observation.

Reserve variables t, t_5, t_k , etc., to denote instants (or points) in TIME . The function mapping events to their observation instants is named T . For any event e , $T\langle e \rangle \in \text{TIME}$ is that point. It is assumed that all events are distinguishable in TIME , so that T is injective. In other words,

$$\text{if } T\langle e_1 \rangle = T\langle e_2 \rangle \text{ then } e_1 = e_2$$

Clocks

Clocks are devices that measure discrete time, usually in a periodic way. A clock C is represented by an sequence of events called *ticks*:

$$C = \{C^0, C^1, C^2, \dots\}$$

Ticks are strictly ordered in time, so that for all $i < j$,

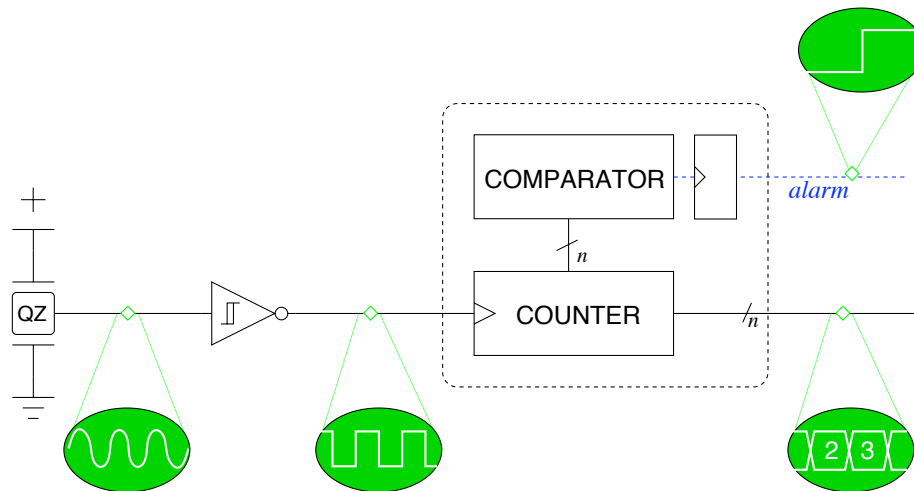
$$T\langle C^i \rangle < T\langle C^j \rangle$$

Just as the variable t is overloaded to denote both real-time instants and event observation times ($T\langle e \rangle$), let us define $C(e) \in \mathbb{N}$ to be the *index of the tick* at which e is observed by clock C . It should be clear that for any event e , $T\langle e \rangle \leq T\langle C^{C(e)} \rangle$; that is, no event can be observed in discrete time before it actually happens.

However, unlike the case in TIME , distinct events can be co-incident in “clock time;” we can have $C(e_1) = C(e_2)$ even if $e_1 \neq e_2$.

Physical Clocks

Physical clocks usually contain an *oscillator* of some kind¹ that generates a continuous sinusoidal voltage wave-form. The wave-form passes through a voltage-thresholding device that generates a digital wave form whose transitions (from low to high voltage, say) are the clock's ticks. This signal forms the *hardware clocking* hardware signal that synchronizes all digital components in its "domain." In effect, each device becomes an observer of the hardware-clock's ticks.



At the programming level, one is not so concerned with the clock-ticks themselves, but instead with how many ticks have transpired. Programs deal with *timers* which are basically digital counters that tally the ticks of the hardware clock. This counter can be programmed to generate an interrupt signal when the tally exceeds a pre-determined value. Every computer has one or more timers of this kind.

Of course, no physical clock keeps perfect time. True oscillators have specified frequency ranges and may differ from their frequency ratings within that range. True frequencies are also subject to fluctuations caused by electro-thermal conditions, and even the age. An oscillator's true frequency may *drift* (or deviate) by 0.00001% to 0.1% from its specified frequency, depending on its quality.

The reciprocal of a clock's frequency (cycles per second) is its *period* or *cycle time*. Technically, these terms refer to the continuous behavior of the oscillator. In discrete-time models, the comparative term *granularity* is also used to characterize the duration between ticks, and a specific cycle is called a *granule*. All these terms are interchangeable when the context is understood. A clock with a higher frequency, hence shorter cycle time, is said to be *fine-grained* relative to a lower-frequency, or *course-grained* clock.

¹The most common kind of oscillator is a quartz crystal, which vibrates in the presence of voltage, generating a fixed-period sine-wave.

The Reference Clock

The following kinds of questions are of interest:

- How accurate is a process’s estimate of global real time?
- How far apart in real time must two events be in order for a process to distinguish them?
- Can two processes observing the same event agree on when it occurred?
- If not, is there a best-case bound on their time estimates?
- Can two or more processes agree on what time it is?

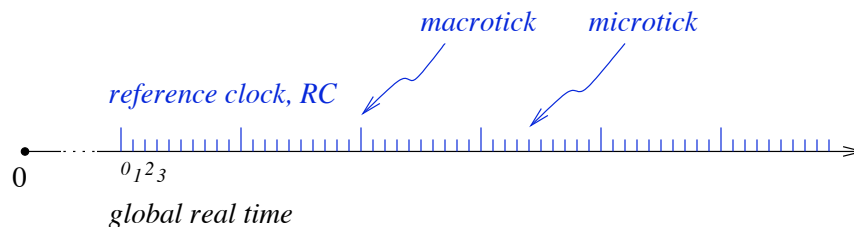
To answer such questions, we need some kind of “ruler” providing a common reference to discrete time and to serve as a mathematical interface between digital time-keeping and global real time. The *reference clock*, called R , is a unique, abstract clock. R is observable externally but not internally, and it keeps perfect calendar time, relative to an International Time Standard. Each granule of R is strictly equal to all others. That is, R ’s frequency is *exactly* $f = \frac{1}{p}$ for some fixed p so that for all $i \in \mathbb{N}$,

$$T\langle R^{i+1} \rangle - T\langle R^i \rangle = p$$

R ’s is sufficiently fine-grained to make all digitization errors negligible, or equivalently, to make all distinct events separable. In current technologies, a frequency of femtoseconds (10^{15} cycles per second) is sufficient because digital *signal events*—transitions between valid voltage levels—require several femtoseconds.

Microticks and Macroticks

The reference clock’s miniscule granularity also allows us to mathematically measure and compare the presumably coarser-grained behavior of physical clocks. Toward this end, a coarser grained clock is superimposed on R , whose ticks represent the ideal behavior of a physical clock. Rather than postulating still another clock, we identify certain ticks of R as *macroticks*. The fine-grained ticks are called *microticks*. Unless the context requires it, the term *tick* means macrotick. So the picture of global time becomes:



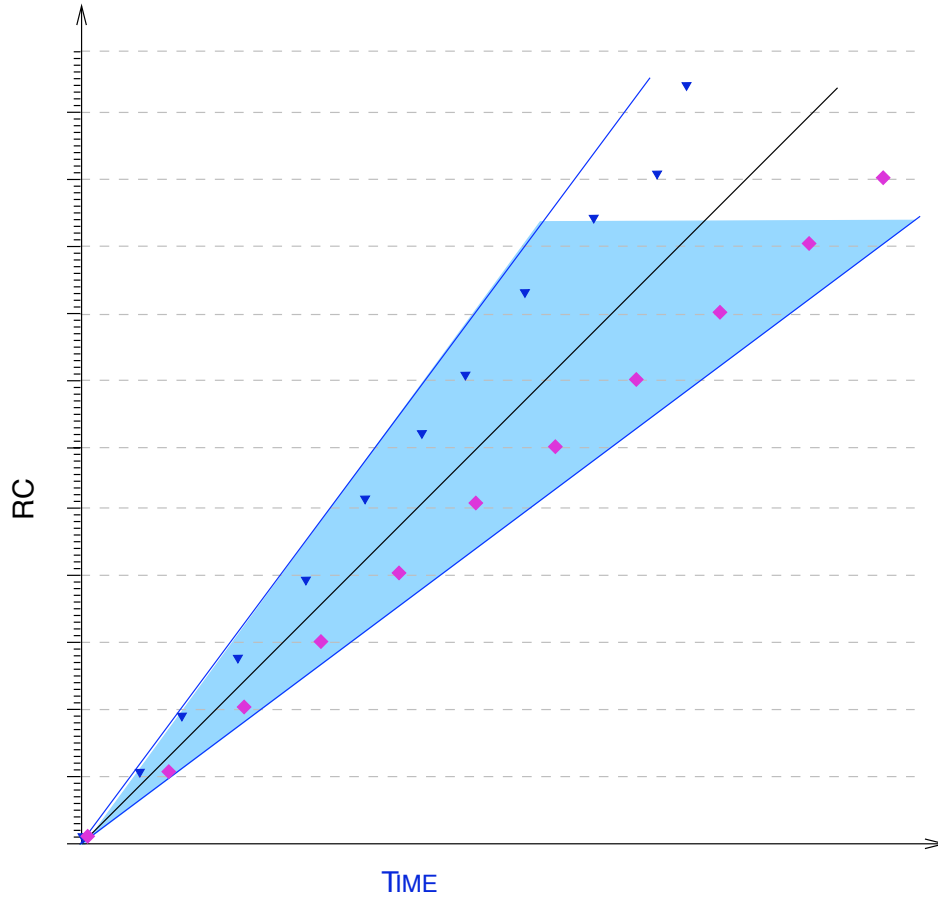


Figure 1: Clock drift

Notions of microticks and macroticks carries to physical clocks, as we shall see in Section , and beyond into levels of software abstraction. At software levels, it is not useful to reason at the physical clock's level of granularity. A single CPU instruction, for example, takes several hardware-clock ticks to interpret. So eventually, a notion of physical macroticks is needed. Similarly, at software levels, a macrotick may correspond to thousands or millions of physical clock ticks, depending on the granularity of software cycles.

Clock Drift

Figure 1 graphs ticks of two physical clocks against real time (horizontal axis) and reference-clock time (vertical axis). Both clocks deviate from the ideal

clock, whose ticks would lie along the diagonal line with slope 1. One is too fast and also fluctuates from tick to tick, when compared to the reference clock. The other is too slow, but appears to be more regular with respect to R . Clock *drift* is its deviation from a specified frequency as measured by R in microticks:

DEFINITION. *The incremental drift of a physical clock at tick i is*

$$\text{drift}(i, C) = \frac{R \langle C^{i+1} \rangle - R \langle C^i \rangle}{n}$$

where n is the nominal number of microticks successive ticks.

C 's incremental drift rate is its drift, normalized to an ideal clock whose incremental drift is 0.

$$\rho(i, C) = |\text{drift}(i, C) - 1|$$

The (accumulated) drift rate, $\rho(C)$, defines a bound on C 's drift over time.

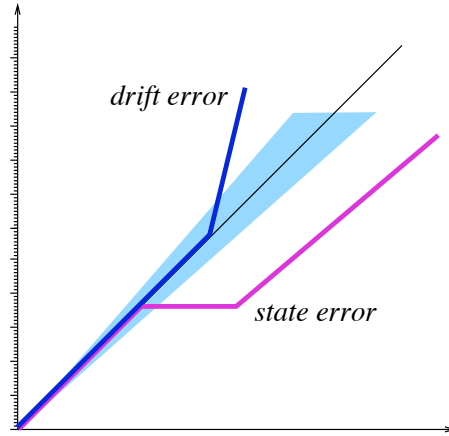
$$\rho(C) = \max\{\rho(i, C)\}_{i \in \mathbb{N}}$$

A physical clock's drift rate within specified environmental conditions is documented in its data sheet. As noted earlier, typical drift rates range from 10^{-2} to 10^{-7} , depending on the quality of the oscillator. Since no physical clock has drift rate 0, any *free running* clock will, over time, drift arbitrarily far from its nominal frequency; that is, that is, no bound can be placed on cumulative drift. Hence, physical clocks must occasionally be *adjusted* to correct for their drift.

Clock Failure Modes

Even an adjusted clock may still *fail*, however. Two distinct *modes* are commonly of concern. A *drift failure* occurs when a clock exceeds its specified drift rate. Drift failure is typically caused by environmental conditions exceeding the specified constraints, or when rate-adjustable clocks (discussed later) are mis-configured. A *state error* occurs when the clock's counter is over-written with an invalid value. Both of these scenarios, illustrated below, are possible effects

of software errors in clock adjustment.



Multiply Clocked Systems

Embedded systems usually contain multiple, separately clocked components. Coordinating component behavior requires that they agree on time at some level of granularity. More specifically, if two components observe the same event, they should agree as to when that event occurred. Similarly, if two components observe two events, they should agree on their temporal order.

Ordinarily², no physical clock can observe the ticks of any other clock, but the reference clock can observe their relative drift, or *offset*.

DEFINITION. *The offset between two physical clocks, C and D , at tick i , is the number of microticks between their corresponding i^{th} ticks:*

$$\text{offset}(i, C, D) = |R\langle C^i \rangle - R\langle D^i \rangle|$$

Given an ensemble (or set) of clocks, $\mathcal{C} = \{C_1, \dots, C_n\}$, the precision of \mathcal{C} at tick i is

$$\Pi(i, \mathcal{C}) = \max\{\text{offset}(i, C_j, C_k) \mid 1 \leq j, k \leq n\}$$

The precision Π of \mathcal{C} over an interval of interest, $[a, b]$, is

$$\Pi = \max \{ \Pi(i, \mathcal{C}) \mid a \leq i \leq b \}$$

Since precision is simply a measure related to relative drift, it grows without bound over time unless the clocks are internally synchronized occasionally. Clock synchronization is the subject of Section ??, later in this chapter.

²Important exceptions to this generality are discussed later in this book, but not in this chapter.

In most applications, the primary concern is precision; that is, how closely the physical clocks agree on the number of ticks that have occurred since time zero. Sometimes, it is also important to know how far the ensemble has drifted in its estimate of global real time.

DEFINITION. A clock's Accuracy at tick i is

$$\text{accuracy}(i, C) = \text{offset}(i, C, R)$$

The accuracy of an ensemble $\mathcal{C} = \{C_1, \dots, C_n\}$ is the maximum accuracy its individual clocks over an interval of interest.

When accuracy is a concern, all physical clocks must also be *externally synchronized* with the reference clock from time to time.

If an ensemble \mathcal{C} is externally synchronized to an accuracy of A then it is also internally synchronized to a precision of $\Pi = 2A$. An internally synchronized ensemble is not necessarily accurate, however.

Time Estimation

We have laid enough groundwork, now, to address the questions posed at the beginning of Section . In particular, let us consider events that are visible³ to the physical clocks and whether they can agree on the global time of such a common observation.

It should be clear that agreement cannot be closer than the precision of the ensemble, so let us impose a coarser granularity of that magnitude. Given an ensemble $\mathcal{C} = \{C_1, \dots, C_n\}$ with precision Π , choose a *resolution* $m \in \mathbb{N}$ such that for each $C_k \in \mathcal{C}$ and for all $i \in \mathbb{N}$,

$$R \langle C_k^{i+m} \rangle - R \langle C_k^i \rangle \leq \Pi$$

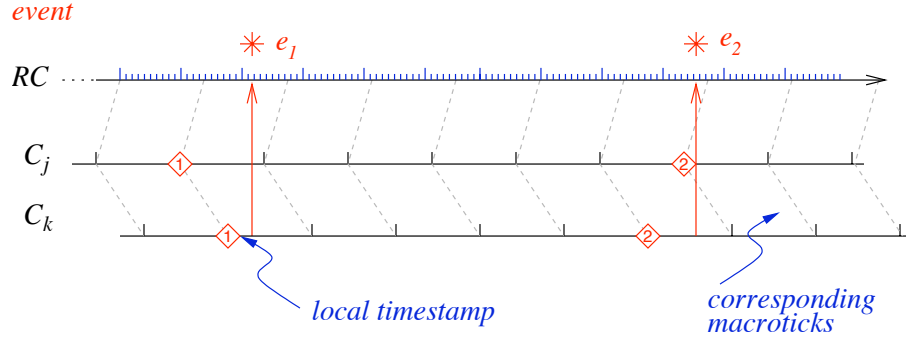
As with the reference clock, call the fine-grained ticks of a physical clock its *microticks* and call every m^{th} microtick a *macrotick*, or simply *tick* if the context allows. A *timestamp* is a local clock's estimate of global time, is measured in (macro)ticks, still denoted by $C \langle e \rangle$. This coarser *global granularity* of macroticks, we can assure that the local timestamps of two clocks observing the same event differ by at most one tick; for any event e

$$| C_j \langle e \rangle - C_k \langle e \rangle | \leq 1$$

This is as close as we can get to clock agreement with respect to a single event. Because physical clocks cannot be perfectly synchronized in dense TIME, for an event to occur between the i^{th} tick of one clock and the $i + 1^{\text{st}}$ tick of another, resulting in different timestamps. In the illustration below, two clocks agree on

³Of course, the clocks themselves don't observe or act on events; they are simply event generators, used by the observers to mark the time of their observations.

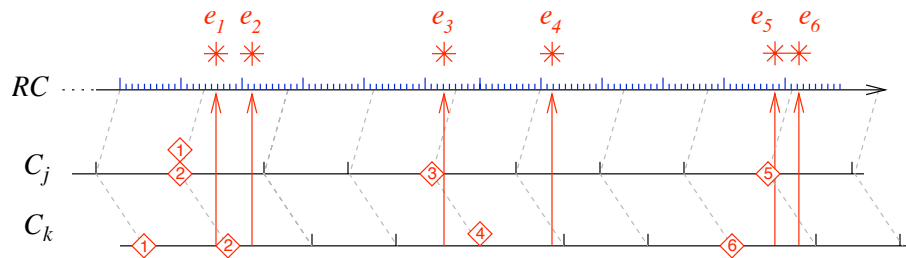
event e_1 and disagree on e_2 . It is of no help to make the granules larger. This is a fundamental disparity.



Event Ordering

It should be apparent that if two events, e_1 and e_2 , occur close enough in time, then a single clock cannot distinguish their order. Similarly, in the illustration below,

- (a) $C_j \langle e_1 \rangle = C_j \langle e_2 \rangle$, but $C_k \langle e_1 \rangle \neq C_k \langle e_2 \rangle$. Clock C_j sees events e_1 and e_2 occurring simultaneously but C_k sees them at different global times.
- (b) $C \langle e_3 \rangle = C \langle e_4 \rangle$, but $R \langle e_4 \rangle - R \langle e_3 \rangle > \Pi$. The *ensemble* $\{C_j, C_k\}$ sees events e_3 and e_4 simultaneous in global time, even though the actual interval between them exceeds the precision of the system.
- (c) $C \langle e_6 \rangle < C \langle e_5 \rangle$, but $R \langle e_5 \rangle < R \langle e_6 \rangle$. The *ensemble* $\{C_j, C_k\}$ sees events e_5 and e_6 in a different temporal order than they actually occur.



Scenarios (b) and (c), above, demonstrate that of the temporal order of two events can be correctly deduced provided the actual time-interval between them is at least two granules of local time. As a consequence of of synchronization *and* digitization errors it is not (always) possible for a multi-clocked *system* to reconstruct event order if the timestamps differ by one tick, even though this is possible for a single local clock.

Measuring Time Intervals

As we have just seen, the synchronization and digitization errors associated with a single event sum to at most $2g$. If we wish to measure the time interval between starting and ending events, e_s and e_e , define *durations*

$$d_{act} = R\langle e_e \rangle - R\langle e_s \rangle, \text{ the actual interval from } e_s \text{ to } e_e$$

$$d_{obs} = \mathcal{C}\langle e_e \rangle - \mathcal{C}\langle e_s \rangle, \text{ the observed interval from } e_s \text{ to } e_e$$

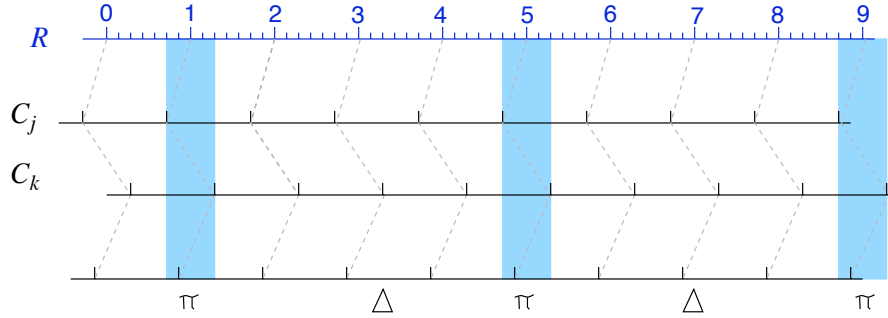
Then if $g > \Pi$ is the global granularity of the system.

$$(d_{obs} - 2g) < d_{act} < (d_{obs} + 2g)$$

At best the actual duration is within a $4g$ time window of what can be observed.

π/Δ Precedence

So far, we have looked at external event discrimination from within a distributed system. Let us now turn this around and consider the external view of events generated from within such a system. Suppose three separately clocked nodes are each to *generate* events at regular intervals, at ticks 1, 5, and 9.



DEFINITION. Let \mathcal{C} be a system with precision Π and EVENT a set of events. Given two durations $\pi < \Pi$ and $\Delta \gg \pi$, \mathcal{C} is said to be π/Δ precedent if, for all $e_1, e_2 \in \text{EVENT}$,

$$|R\langle e_1 \rangle - R\langle e_2 \rangle| \leq \pi \text{ or } |R\langle e_1 \rangle - R\langle e_2 \rangle| > \Delta$$

An external observer seeing two events separated in time by either π or less should disregard their temporal order. Since they occur within the precision of the system, it should be assumed that they were intended to be coincidental in discrete time. If the separation is Δ or greater, it should be assumed that they were intended to occur at different times.

The ideal precedence is $0/\Delta$, where it is supposed that the nodes of \mathcal{C} are perfectly synchronized. The value of Δ needed to recover the order of two

events depends on the global granularity, g , of a distributed system's time base. Verissimo⁴ shows that a system must have $0/3g$ precedence in order to recover the temporal order of two events. Let $\delta(e_1, e_2)$ denote the minimum difference in timestamps between two non-simultaneous events:

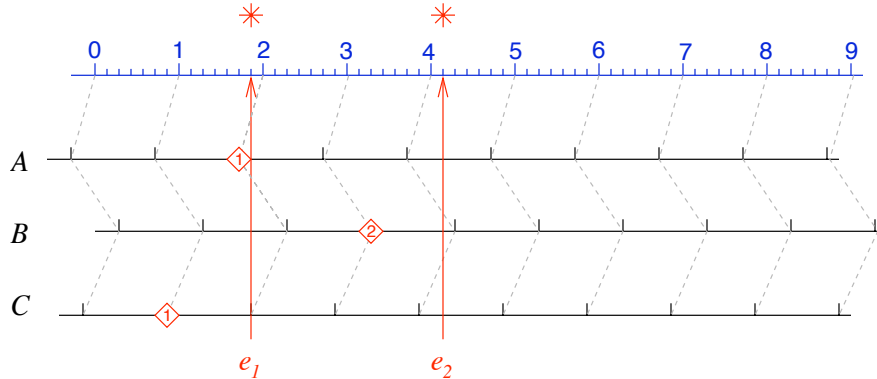
$$\delta(e_1, e_2) = \min\{ |C_j \langle e_1 \rangle - C_k \langle e_2 \rangle| \}_{1 \leq j, k \leq n}$$

<i>precedence</i>	δ	<i>order?</i>
0/1	$\delta(e_1, e_2) \geq 0$	no
0/2	$\delta(e_1, e_2) \geq 1$	no
0/3	$\delta(e_1, e_2) \geq 2$	yes
0/4	$\delta(e_1, e_2) \geq 3$	yes

As discussed in Section , a difference of at least two ticks is needed to properly order two events, so in a $0/3g$ precedent system order can always be recovered. This means that a distributed system must be designed to assure a 3-tick period of silence, at least, between two *internally generated* events whose temporal order matters. Of course, the system has no control over the lapses between externally generated events.

Agreement Protocols

Consider the scenario below in which events e_1 and e_2 occur $2\frac{1}{2}$ granules apart. Node C observes e_1 at time 1; Node A observes the same event at time 2. Node B observes e_2 at time 3 and reports this observation to A and C . Node C calculates a two-tick interval between e_1 and e_2 and surmises that e_1 occurred *strictly before* e_2 . Node A calculates a one-tick interval and concludes that e_1 and e_2 *cannot* be ordered.



To resolve *inconsistent* views of event ordering, the nodes must execute an *agreement protocol*, in which they first exchange information about their re-

⁴P. Verissimo. Ordering and Timeliness Requirements of Dependable Real-Time Programs. *Real-Time Systems* 7(3):105–128 (1994).

spective local views, and then individually decide a consistent—not necessarily correct—view. In this example, that protocol could be either:

- if some node witnesses an event ordering, then all nodes agree that they are ordered, or
- if some node fails to order the events, then all nodes agree that they cannot be ordered.

QUESTION: *Is it possible for two nodes to agree that two events are ordered, but disagree about which event occurred first?*

Agreement algorithms can be quite expensive, owing to the overhead of information exchange, especially when other considerations such as faulty behavior are taken into account. The dominant cost, usually, is not computational overhead, but the delay imposed by executing the protocol.

Clustering

A common way to ameliorate agreement costs is to organize systems hierarchically into *clusters* so that intra-cluster coordination such as agreement protocols can be more localized. It is reasonable to infer that such localization will lower the precision of inter-cluster coordination.

Suppose we have two clusters, \mathcal{A} and \mathcal{B} which both have precision Π . If nodes $A_1, A_2 \in \mathcal{A}$ generate events at the same tick i , they may be as much as Π apart in global time, so the observing cluster \mathcal{B} should never impose a temporal order on two such events. Conversely, \mathcal{B} *should* always order two events that occur in separate cluster-wide ticks.

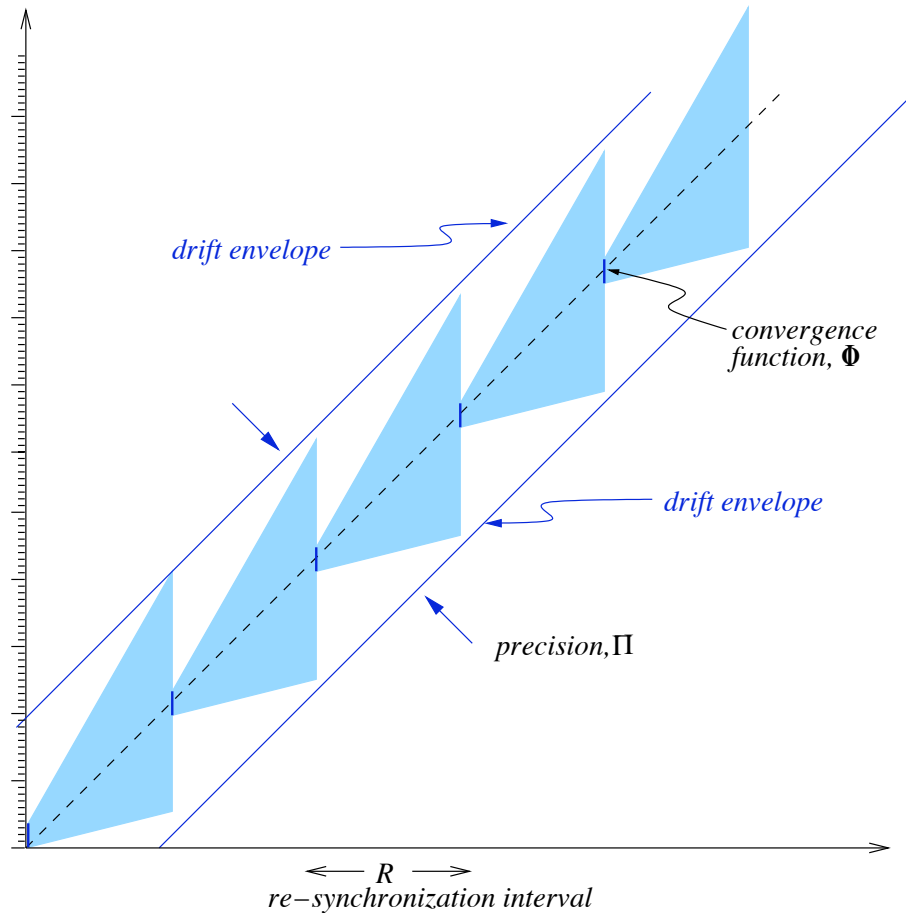
Let g be the cluster granularity and suppose both \mathcal{A} and \mathcal{B} are $1g/3g$ precedent. Two events generated by \mathcal{A} in the same local macrotick may actually occur in two successive ticks of global time, and hence, might be observed by \mathcal{B} with timestamps that differ by *two*. Likewise, two \mathcal{A} -events differing by three local macroticks—and thus potentially temporally ordered—might be observed by \mathcal{B} with timestamps that differ by *two*. In other words, without an agreement protocol, \mathcal{B} cannot always distinguish properly ordered from coincidental events emanating from \mathcal{A} .

This ambiguity is resolved by making \mathcal{A} $1g/4g$ -precedent, requiring at least four ticks of quiescence between events that are to be externally ordered.

Internal Synchronization

Since the offset between free-running clocks is unbounded, it is necessary periodically to re-synchronize clocks in a distributed system. In this context, the *precision*, Π of a system is the maximal *offset* between any two physical clocks.

The goal is to determine a (re-)synchronization interval, R , at which to adjust the physical clocks to keep within a *drift envelope* bounding their offset.



Recall (Sec. ?? that the *drift rate*, ρ is a measure of deviation from a specified microtick rate. The system's drift offset, denoted Γ , is given by

$$\Gamma = 2\rho R$$

The clock correction, or *convergence function*, Φ defines the abound on physical clock offset immediately after they have been adjusted. It should be clear that in a physical system Φ is not zero. The bounds on Φ are

$$0 \leq \Phi \leq \Pi - \Gamma$$

This only says that convergence must be "close enough" that the clocks do not drift out of their envelope in the next synchronization interval, or equivalently, we must make R small enough to keep the clocks inside the envelope.

We shall see later that a resilient clock synchronization agreement protocol is rather expensive. For now, we look at some of the simplest forms.

Master Clock

A *master clock* is a physical clock used as an internal reference clock. It is entrusted with system time-keeping. Periodically, the master clock broadcasts the time to the entire system. On observing this message, all “slave” clocks adjust themselves to agree, within system precision, with the master time.

Typically, the master clock does not tell the system literally what time it is. Instead, it sends *sync* messages. . .

MORE TO COME. . .

Distributed Agreement Protocol

MORE TO COME. . .