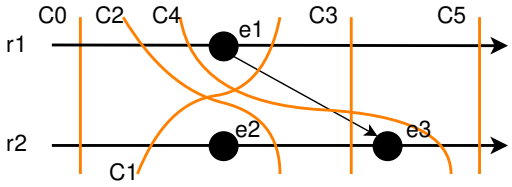


**Figure 2:** Space/time diagram and possible cuts of a computation.

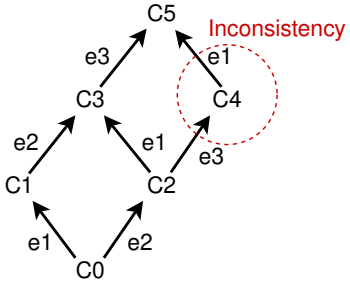


**Figure 3:** Lattice of global states of the computation depicted in Fig. 2.

(with respect to $\rightarrow$ they are not ordered). Hence, it cannot be determined which sequence of global states occurred in the computation, as long as the cut respects the causality relation $\rightarrow$. Cut $c_4$ is one that violates $\rightarrow$ in that events $e_2$ and $e_3$ are contained in the global state while $e_1$ is not. This cannot have happened since $e_3$ is the effect of $e_1$, i.e., if $e_3$ is contained in the cut, then $e_1$ also must be. This is the basis of the definition of a *consistent cut*.

### 2.6. Realtime

In systems where real-time clocks are available, it may be possible to order two events $e_1$ and $e_2$ in Fig. 2 by comparing the real-time readings of when they occurred. For an event $e$ we denote by $rt(e)$ the real-time clock reading when $e$ occurred. Formally, $rt$ is a function mapping the set of events to the time domain $T$. For simplicity and without loss of generality, we equate $T$ with the set of natural numbers. Using $rt$, it is possible to transform the partial order $\rightarrow$ into a total order by ordering every event $e \in E$ using $rt(e)$.

### 2.7. Snapshots

The effects of events are possible value changes in the memory regions. Following the notation of Vömel and Freiling (2012), we define the set of all possible values of a memory region as $V$. The contents of the memory regions can then be formalized as a function $m : R \times T \rightarrow V$ that returns the value of a specific memory region at a specific point in time. Function $m$ encodes a form of *ground truth* of what values the memory contained at any specific time.

Informally, a *snapshot* is a copy of all memory regions. Since individual memory regions might be copied at different points in time, we formalize a snapshot as a function $s : R \rightarrow V \times T$, i.e., for every memory region we store the value and the time this value was copied from memory. We denote by $s(r).v$ the value stored for region $r$ in snapshot $s$ and by $s(r).t$ the corresponding time. For example, if $s(r_1) = (15, 3)$ then memory region $r_1$ was copied at time $s(r_i).t = 3$ with a value of $s(r_i).v = 15$. Snapshots correspond to cuts through the space-time diagram of a computation.

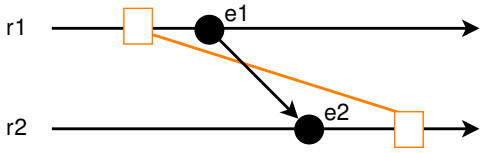


**Figure 4:** Causally inconsistent snapshot.

Taking a snapshot of a computation means to copy the current values from memory regions into the snapshot, but a snapshot does not contain any references to events that have happened. To be able to formally connect events in and snapshots of a computation, we introduce one additional notation: For a real-time value $t$ and a memory region $r$ we denote by $event(r, t)$ the *most recent event* that happened on memory region $r$ at a time before or equal to $t$. Formally, *event* is a function $event : R \times T \rightarrow E$. If $event(r, t) = e$ then $rt(e) \leq t$ and there exists no other event on $r$ that happened between $rt(e)$ and $t$.

Technically, the set of events contained in the cut corresponding to a snapshot $s$ consists of all events that lie to the left of $event(r, s(r).t)$ (including the event itself).

## 3. Defining Atomicity

Intuitively, atomicity is a notion to characterize the degree of freedom of signs of concurrent activity. High atomicity therefore attempts to bound the effects that arise from an observation taking place concurrently to a computation.

### 3.1. Causal consistency

The original definition of atomicity introduced by Vömel and Freiling (2012) is based on the causal dependency relation $\rightarrow$ between events. It states that the set of events derived from a snapshot corresponds to a consistent cut. The rationale behind this definition was that snapshots should respect causality, i.e., for each effect the snapshot contains its cause. The definition rules out any inconsistent cuts as allowed snapshots. Such an example is depicted in Fig. 4 where two events $e_1$ and $e_2$ occurred on region $r_1$ and $r_2$ respectively and $e_1 \rightarrow e_2$. A snapshot that contains the contents of $r_1$ before $e_1$ happened and the contents of $r_2$ after $e_2$ happened is causally inconsistent because the change introduced by $e_1$ that caused $e_2$ is missing.

Vömel and Freiling (2012) argued that snapshots should at least be consistent with causality because causally inconsistent snapshots clearly cannot have happened. Unfortunately, many software-based snapshot approaches for RAM do not produce even causally consistent snapshots.

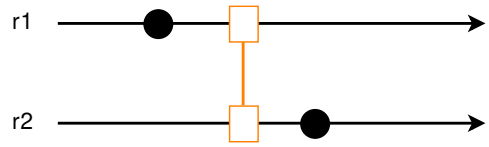


**Figure 5:** When an instantaneous snapshot is taken, all memory regions are copied at the same time.

## 3.2. Instantaneous consistency

Causally consistent snapshots guarantee that snapshots respect causal relationships. However, causal consistency is a notion defined for asynchronous distributed system, i.e., systems where no notion of real-time exists and time is reduced to causality. In such systems, events can be reordered along the sequential timeline if causal relationships are respected. Therefore, *every* consistent global state is a state that the computation *potentially* could have passed through. In practice, and in particular in those systems that we focus on here (smartphones, PCs, servers), often a notion of real-time exists that allows to narrow down the set of consistent global states that *actually* have happened (Stoller, 2000).

Based on these insights, we now define an idealistic (and much stricter) consistency criterion based on the time at which the memory regions are copied. The notion of *instantaneous consistency* formalizes the idealistic intent of snapshots in which all memory regions are copied at exactly the same time.

**Definition 1** (instantaneous consistency)**.** *A snapshot $s$ satisfies* instantaneous consistency *iff all memory regions in $s$ were acquired at the same point in time. Formally:*

$$\forall r, r' \in R : s(r).t = s(r').t$$

If a snapshot satisfies instantaneous consistency, we say that the snapshot *is* instantaneous. Obtaining instantaneous snapshots is possible if the system of which the memory contents are extracted can be paused, for example when the main memory of a virtual machine is dumped. An example of an instantaneous snapshot is depicted in Fig. 5.

From a forensic point of view, it is desirable that a snapshot is instantaneous because it resembles something that is easy to understand and "obviously" free of any problems of concurrency. This aspect is important for legal proceedings in which doubts on the way evidence was gathered can severely degrade its evidential value.

## 3.3. Quasi-instantaneous consistency

Taking instantaneous snapshots usually requires freezing the system from which memory is copied, at least this is the case for systems where no inherent (hardware) mechanism exists to copy all memory regions at the same time. So taking instantaneous snapshots in practice is hard, if not impossible.

We therefore define a slightly weaker criterion that captures the main ideas of instantaneous consistency while allowing to take snapshots without freezing the system. We call this *quasi-instantaneous consistency*.

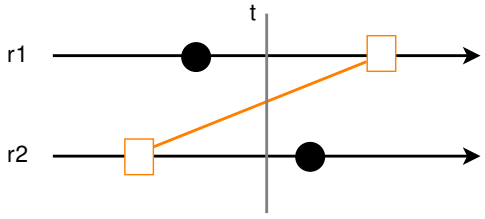


**Figure 6:** When a snapshot satisfies quasi-instantaneous consistency, a point in time can be found at which the contents of the memory regions in the snapshot coexisted in the copied main memory. The same result would have been achieved with an instantaneous snapshot at time $t$.

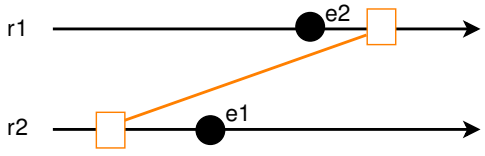


**Figure 7:** A snapshot that is not quasi-instantaneously consistent if $e_1$ and $e_2$ modify the values of $r_1$ and $r_2$, respectively.

**Definition 2** (quasi-instantaneous consistency)**.** *A snapshot $s$ satisfies* quasi-instantaneous consistency *iff the values in the snapshot could have also been acquired with an instantaneous snapshot $s'$. Formally:*

$$\exists s' : (\forall r, r' \in R : s'(r).t = s'(r').t) \wedge \\ \forall r \in R : s'(r).v = s(r).v$$

The above definition does not require that the snapshot *is* taken instantaneously but that it *could* have been taken instantaneously, i.e., that the values of all memory regions in the snapshot were coexistent in memory at (at least) one point in time during the acquisition.

Fig. 6 shows an example of a snapshot which is quasi-instantaneous. In this example a point in time can be found at which the contents of the two memory regions in the snapshot were coexistent in memory. When such a point in time cannot be found, the snapshot is not quasi-instantaneous. Assuming modifying events, Fig. 7 shows an example for this case. Because of the order of the events $e_1$ and $e_2$ and the time points at which the memory regions were added to the snapshot, the snapshot contains values that were never coexistent in main memory. In this case it is impossible to find a time at which a snapshot containing the same values could have been taken instantaneously.

## 3.4. Relations between the consistency definitions

Instantaneous consistency is the strongest concept of the presented consistency definitions. If all memory regions can be copied at the same time, no inconsistencies can arise due to concurrent activity. Therefore, instantaneous consistency implies quasi-instantaneous consistency.

The relation between quasi-instantaneous and causal consistency is slightly less apparent. We first argue that

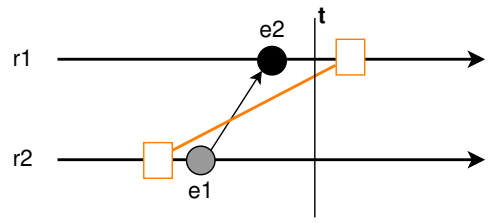


**Figure 8:** If $e_1$ is an event that does not change the memory contents, then the snapshot is quasi-instantaneously consistent but not causally consistent. If $e_1$ and $e_2$ are modifying events, then the snapshot is neither quasi-instantaneously nor causally consistent.

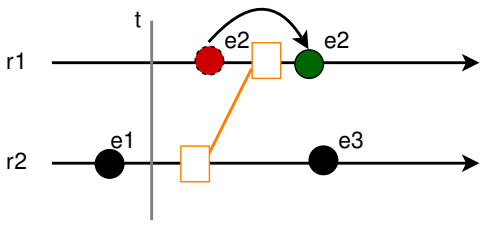


**Figure 9:** After the acquisition has been started at time t, no changes to memory regions that have not been copied yet are allowed. Once a region has been copied its content may be changed by events.

a causally consistent snapshot is not necessarily quasi-instantaneously consistent. To see this, consider the computation in Fig. 7 and note that the events $e_1$ and $e_2$ are independent of each other. Therefore, any snapshot of this computation is causally consistent. However, if $e_1$ and $e_2$ are modifying events, the snapshot is not quasi-instantaneously consistent. So causal consistency does not generally imply quasi-instantaneous consistency.

But what about the inverse question, i.e., is every quasi-instantaneous snapshot also causally consistent? Interestingly, the answer to this question depends on the nature of events that determine causal consistency. To see this, consider the computation in Fig. 8 which is similar to the one depicted in Fig. 7 but where events $e_1$ and $e_2$ have a causal relationship. If neither $e_1$ nor $e_2$ are modifying events then the values stored in memory do not change and so any snapshot would be quasi-instantaneous, also the one depicted in Fig. 8. So in this case, a snapshot might be quasi-instantaneously consistent but still causally inconsistent. But even if we only have modifying events, the changes of $e_1$ or $e_2$ could be reverted and the resulting quasi-instantaneously consistent snapshot might again not be causally consistent. But if we assume that we only have uniquely modifying events, this cannot happen anymore.

**Proposition 1.** *If all events are uniquely modifying, then any quasi-instantaneously consistent snapshot is also causally consistent.*

*Proof.* Let $s$ be a snapshot satisfying quasi-instantaneous consistency. From the definition of quasi-instantaneous consistency follows that there exists an instantaneous snapshot $s'$ that contains the same values as $s$ for every memory region. Since $s'$ is instantaneous, it is also causally consistent. But since all events are uniquely modifying, no events can have happened between $s'$ and $s$. Therefore, $s$ is also causally consistent. □

As observed by Pagani et al. (2019), causal consistency is very permissive but appears to be the smallest common denominator of any acceptable quality measure of atomicity. However, it is too permissive to be easily attainable. Instantaneous consistency, the ideal notion of atomicity, is too strong. Quasi-instantaneous consistency is an intermediate definition that does not need to halt the system but still can express a similar level of instantaneousness. It is close in spirit to Pagani et al.'s concept of *time consistency*, which is satisfied "if there was a point in time during the acquisition process in which the content of those pages co-existed in the memory of the system" (Pagani et al., 2019).

## 4. Achieving Consistency

One possibility to achieve quasi-instantaneous consistency of snapshots created on a running system is to ensure that, after the acquisition process has been started, no memory content will be modified before it has been copied. This method is known as *copy-on-write* in the area of systems software. It is rather easy to see that a snapshot created using this technique satisfies quasi-instantaneous consistency, because any state changes occurring after the start of the acquisition are not included into the memory snapshot. Therefore, the contents in the snapshot are equal to those the memory regions contained at the start of the acquisition.

An example can be seen in Fig. 9: Because $e_2$ would have been executed on memory region $r_2$ after the start of the acquisition process but before it was copied, the event is interrupted and the region is copied first. Afterwards the event can be executed.

Manipulating the page table entries is a convenient method to do this. By taking away the write permission of all page tables entries, trying to change the page will result in a page fault that can be handled accordingly. But the system of which the memory snapshot is created should not be manipulated to such a great extent. Therefore, instead of manipulating the page tables of the operating system, a hypervisor can be used. By taking away the write permissions of the guest pages on the hypervisor level, write accesses will cause an exit to the hypervisor. Then the page can be copied and the write permission for the page can be turned on again. Over the last years the technical means to implement the technique changed, as can be seen in the works of Martignoni, Fattori, Paleari and Cavallaro (2010); Yu, Qi, Lin, Zhong, Li and Guan (2012) and Kiperberg, Leon, Resh, Algawi and Zaidenberg (2019). As it cannot always be expected that a system is already virtualized, methods for the "on the fly" virtualization of a system have