White Paper

Deterministische Echtzeitausführung auf Windows-Systemen

Häufige Missverständnisse und ihre Konsequenzen

Zusammenfassung

Der Begriff „harte Echtzeit“ wird in der industriellen Informatik häufig verwendet, jedoch oft auf Messungen von Latenz oder Systemauslastung reduziert. In vielen Fällen werden statistische Beobachtungen – wie durchschnittliche Latenz, Jitter oder Benchmark-Ergebnisse – als Nachweis für Echtzeitfähigkeit präsentiert. Diese Kennzahlen beschreiben die Leistung, belegen jedoch kein deterministisches Verhalten.

Für Systeme, in denen das Verpassen einer Deadline einen Fehler darstellt, ist eine solche Argumentation unzureichend. Harte Echtzeitfähigkeit ist keine Frage von Geschwindigkeit oder Optimierung, sondern von architektonischer Kontrolle über alle zeitkritischen Ausführungspfade. Dieses Dokument untersucht typische Missverständnisse, die entstehen, wenn statistisches Denken auf deterministische Anforderungen angewendet wird, und leitet daraus die architektonischen Konsequenzen ab, die erforderlich sind, um strikte Zeitgarantien auf Windows-basierten Systemen einzuhalten.

1. Geringer Jitter beweist keine Deterministik

Jitter wird häufig als Indikator für Echtzeitfähigkeit interpretiert. Er beschreibt die Schwankung zwischen der minimalen und maximalen beobachteten Latenz innerhalb eines bestimmten Messintervalls. Obwohl geringer Jitter auf stabile Leistung unter bestimmten Bedingungen hinweisen kann, bleibt er eine statistische Eigenschaft.

Die zentrale Einschränkung besteht darin, dass Jitter sowohl von der Beobachtungsdauer als auch von der Abdeckung der Systemzustände abhängt. Eine Messung erfasst nur einen Teil aller möglichen Ausführungspfade. Wenn bestimmte Systemzustände – wie spezielle Interruptmuster, Treiberinteraktionen oder Ressourcen-

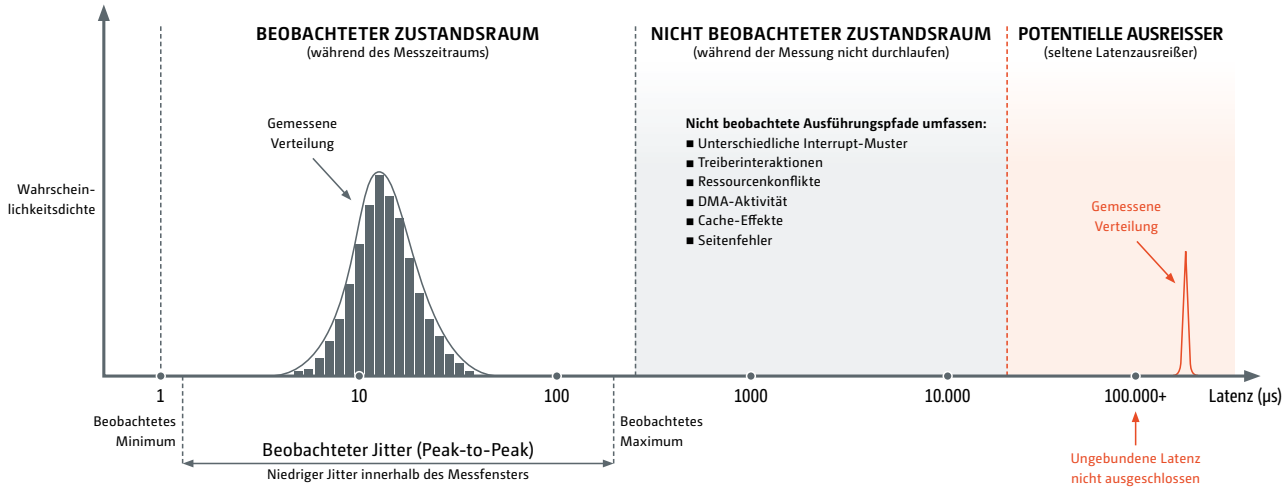
konflikte – während des Beobachtungszeitraums nicht auftreten, bleiben ihre zeitlichen Auswirkungen ungemessen.

Aus Systemsicht wächst die Anzahl möglicher Zustände kombinatorisch durch das Zusammenspiel von Hardwareereignissen, Scheduling-Entscheidungen und parallelen Aktivitäten. Eine vollständige Beobachtung ist daher nicht praktikabel. Ein niedriger beobachteter Jitter schließt deshalb nicht aus, dass außerhalb des gemessenen Bereichs deutlich höhere Latenzen auftreten können.

Deterministische Systeme werden nicht durch enge statistische Verteilungen definiert, sondern durch das Fehlen unbegrenzter Variationen. Jitter kann Variabilität beschreiben, aber nicht nachweisen, dass diese begrenzt ist.

Bild 1: Geringer Jitter bedeutet keine Deterministik

Die gemessene Latenz bildet nur einen Teil möglicher Systemzustände ab.



Kernaussage: Geringer beobachteter Jitter weist auf stabiles Verhalten unter den getesteten Bedingungen hin. Er beweist nicht, dass unter anderen (nicht beobachteten) Systemzuständen keine höhere Latenz auftreten kann.

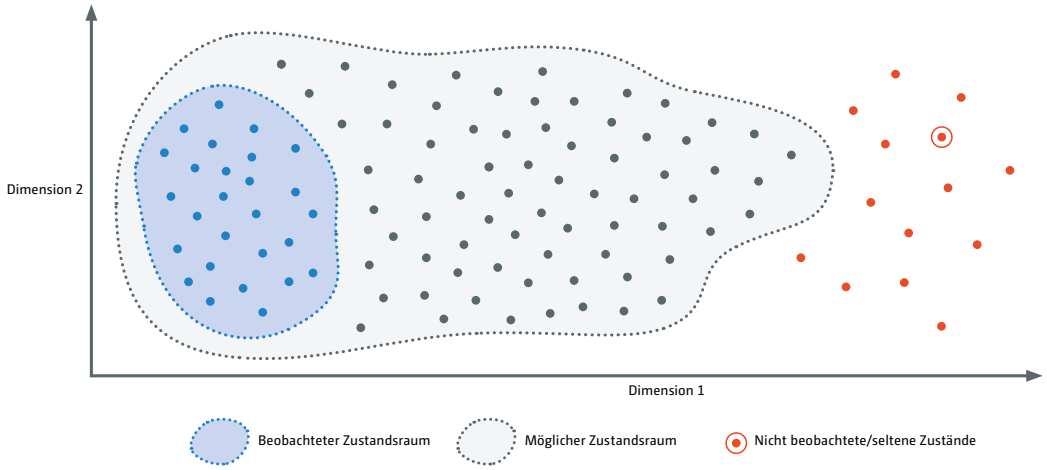
2. Gemessene Maximalwerte sind keine Worst-Case-Grenzen

Eine gemessene maximale Latenz stellt den schlechtesten beobachteten Fall unter bestimmten Testbedingungen dar. Sie ist jedoch keine echte Worst-Case-Grenze, solange nicht nachgewiesen werden kann, dass keine höhere Latenz möglich ist.

Die Bestimmung einer Worst-Case-Grenze erfordert, dass alle zeitrelevanten Mechanismen bekannt und kontrolliert sind. Dazu gehören Scheduling-Entscheidungen, Interrupt-Behandlung, Speicherzugriffsverhalten sowie Interaktionen mit Hardware und Treibern. Wenn eine dieser Komponenten Verzögerungen verursachen kann, die nicht strikt begrenzt sind, bleibt das Zeitverhalten des Systems offen.

In allgemeinen Betriebssystemen werden viele dieser Mechanismen von asynchronen Ereignissen und gemeinsam genutzten Ressourcen beeinflusst. Der resultierende Zustandsraum ist faktisch unbegrenzt, und eine vollständige Untersuchung ist nicht möglich. Messungen erfassen – unabhängig von ihrer Dauer – immer nur einen Teil dieses Zustandsraums.

Bild 2: Beobachteter vs möglicher Zustandsraum



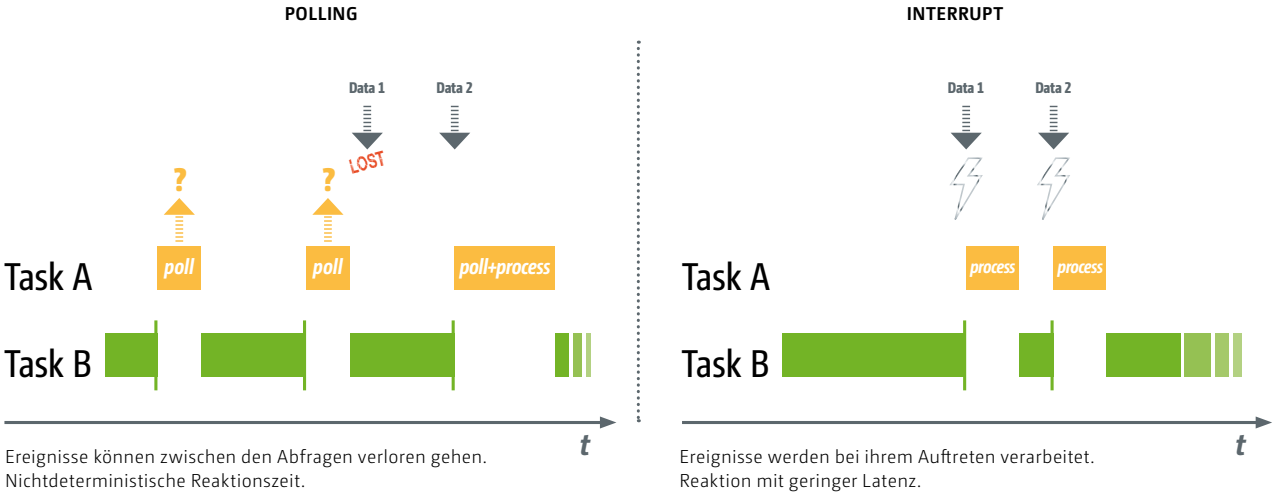
Ein beobachteter schlechtester Fall zeigt lediglich, dass während der Tests kein schlechteres Verhalten auftrat. Ein echter Worst Case hingegen ist eine nachgewiesene obere Grenze, die aus architektonischen Einschränkungen abgeleitet wird.

3. Synthetische Last repräsentiert keine realen Systeme

Synthetische Lasttests werden häufig eingesetzt, um Echtzeitverhalten durch anhaltende Rechenlast zu validieren. Diese Tests basieren typischerweise auf vorhersehbaren, CPU-gebundenen Workloads wie engen Schleifen oder Burn-in-Programmen. Obwohl sie eine hohe Auslastung erzeugen können, bilden sie die zeitliche Struktur realer Systeme nicht nach.

In der Praxis wird das Zeitverhalten eines Systems von asynchronen Ereignissen dominiert. Hardware-Interrupts treten unvorhersehbar auf, Treiber erzeugen variable Ausführungspfade, und Direct Memory Access ermöglicht parallele Datenbewegungen außerhalb der CPU-Kontrolle. Diese Mechanismen interagieren mit Scheduler und Speichersubsystem auf eine Weise, die durch deterministische Schleifen-Workloads nicht erfasst wird.

Bild 3: Vergleich zwischen polling-basierter und interruptgesteuerter Ausführung



Die Interrupt-Behandlung verstärkt diesen Effekt zusätzlich. Interrupts lösen verzögerte Verarbeitung aus, beeinflussen das Cache-Verhalten und können die Ausführung an nichtdeterministischen Punkten unterbrechen. Die resultierende Latenz hängt vom Systemzustand und der Reihenfolge der Ereignisse ab – nicht nur von der CPU-Auslastung.

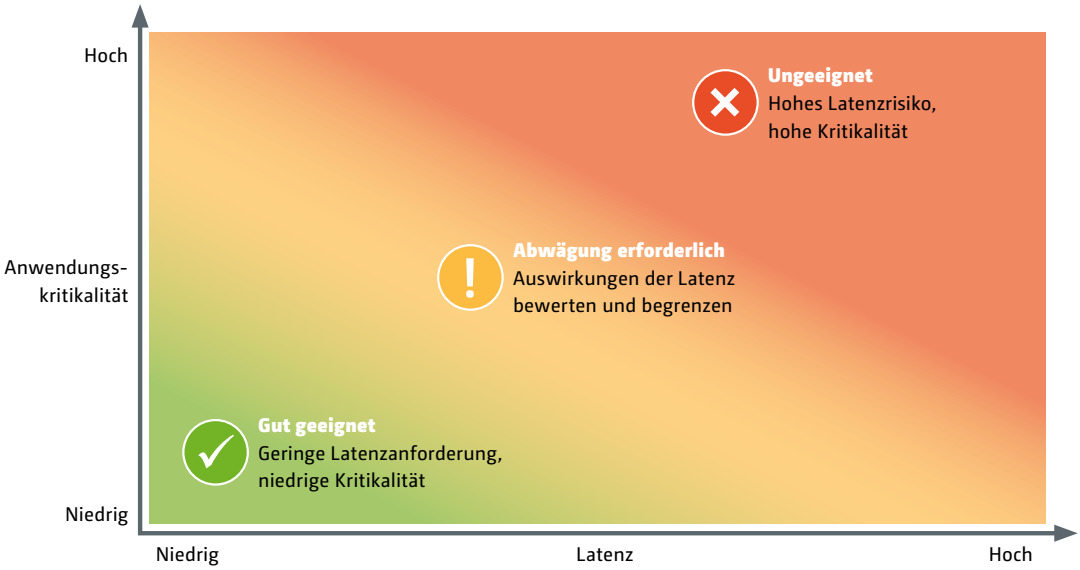
Synthetische Tests schließen daher wesentliche Quellen von Nichtdeterminismus aus und können Echtzeitverhalten unter realistischen Bedingungen nicht nachweisen.

4. Priorität bedeutet nicht Kontrolle

Ein häufig genannter Mechanismus zur Erreichung von Echtzeitverhalten auf Windows ist die Verwendung erhöhter Scheduling-Prioritäten, insbesondere REALTIME_PRIORITY_CLASS. Diese Klasse ermöglicht es Prozessen, oberhalb aller normalen User-Mode-Prioritäten zu laufen und Interferenzen durch andere Anwendungen zu reduzieren. Sie arbeitet jedoch vollständig innerhalb des User-Mode-Scheduling-Modells und verändert das grundlegende Verhalten des Systems nicht. Threads in REALTIME_PRIORITY_CLASS unterliegen weiterhin der Kernel-Mode-Ausführung. Interrupt Service Routines und Deferred Procedure Calls können sie jederzeit unterbrechen – unabhängig von ihrer Priorität. Diese Mechanismen werden von Hardware und Treibern gesteuert und lassen sich aus dem User Space nicht kontrollieren.

Auch die Speicherverwaltung bringt zusätzliche Unsicherheiten mit sich. Page Faults, Kernel-Operationen und Cache-Effekte können die Ausführung auf Arten verzögern, die im User Mode nicht vollständig begrenzt werden können. Der Windows-Scheduler selbst ist auf Reaktionsfähigkeit und nicht auf Determinismus ausgelegt. Er erzwingt keine strikten oberen Latenzgrenzen. REALTIME_PRIORITY_CLASS verbessert daher typische Latenzwerte, beseitigt jedoch nichtdeterministische Einflüsse nicht. Sie bietet Bevorzugung, aber keine Kontrolle, und kann deshalb keine harte Echtzeit garantieren.

Bild 4: Zusammenhang zwischen Latenz und Anwendungskritikalität



5. Numerische Latenzwerte sind kontextabhängig

Latenzwerte werden oft isoliert betrachtet, als wäre eine ausreichend kleine Zahl bereits ein Nachweis für Echtzeitfähigkeit. In der Praxis hängt ihre Bedeutung jedoch vollständig vom Systemkontext und der Fehlersemantik ab.

In sicherheitskritischen Anwendungen wird akzeptables Zeitverhalten nicht durch Durchschnittsleistung bestimmt, sondern durch die Konsequenzen einer verpassten Deadline. Eine Latenz, die für Multimedia-Verarbeitung oder Benutzerinteraktion akzeptabel ist, kann in Motion Control oder industrieller Automatisierung bereits einen Fehler darstellen. Derselbe numerische Wert kann daher – abhängig von der Anwendung – entweder vernachlässigbar oder unzulässig riskant sein.

Diese Unterscheidung ist insbesondere in zertifizierten Umgebungen wichtig. Regulatorische Anforderungen bewerten, ob das zeitliche Verhalten unter allen zulässigen Bedingungen begrenzt bleibt – nicht, ob das System unter typischer Last gut performt. Ein System mit niedriger durchschnittlicher Latenz, aber unbegrenztem Worst-Case-Verhalten, kann daher trotz hervorragender Benchmark-Ergebnisse deterministische Anforderungen verfehlen.

Latenzwerte beschreiben beobachtetes Verhalten unter bestimmten Bedingungen. Für sich allein liefern sie keine Aussage über Garantien, Begrenztheit oder Fehlerbehandlung.

6. Harte Echtzeit entsteht nicht durch Optimierung

Optimierung verbessert den Durchsatz, reduziert die durchschnittliche Ausführungszeit und kann die beobachtete Latenz deutlich verringern. Diese Verbesserungen können die praktische Nutzbarkeit unter Last erhöhen, beseitigen jedoch nicht die zugrunde liegenden Quellen von Nichtdeterminismus in einem allgemeinen Betriebssystem.

Interrupt-Timing, Scheduler-Verhalten, Speicherverwaltungsaktivitäten und Treiberinteraktionen bleiben asynchron und nur teilweise kontrollierbar. Optimierung kann die Wahrscheinlichkeit von Deadline-Verletzungen reduzieren, beweist jedoch nicht, dass solche Verletzungen unmöglich sind.

Diese Unterscheidung wird oft verschleiert, weil hochoptimierte Systeme während Tests stabil erscheinen und statistisch beeindruckende Ergebnisse liefern können. Deterministisches Verhalten wird jedoch nicht dadurch definiert, wie selten eine Deadline verpasst wird, sondern dadurch, ob die Architektur unter allen gültigen Betriebsbedingungen begrenztes Verhalten garantiert.

Harte Echtzeitfähigkeit kann daher nicht allein durch Tuning erreicht werden. Sie erfordert architektonische Kontrolle über zeitrelevante Mechanismen, einschließlich Interrupt-Behandlung, Scheduling-Verhalten und Ressourcenbesitz. Optimierung kann ein deterministisches System verfeinern, aber sie kann eine nichtdeterministische Architektur nicht deterministisch machen.

7. Langfristige Stabilität bedeutet keine Garantie

Lange Phasen stabilen Betriebs werden oft als Beweis dafür interpretiert, dass ein System effektiv deterministisch sei. In der Praxis liefern solche Beobachtungen jedoch nur empirisches Vertrauen auf Basis bisher aufgetretener Betriebsbedingungen.

Komplexe Systeme enthalten eine große Anzahl möglicher Ausführungszustände, die aus Hardwareereignissen, paralleler Softwareaktivität, Scheduling-Entscheidun-

gen und externen Eingaben entstehen. Bestimmte Deadline-Verletzungen können daher nur unter seltenen Zustandskombinationen auftreten, die selbst bei langen Tests möglicherweise nie erscheinen.

Lange Laufzeiten ohne beobachtbare Fehler beweisen nicht das Fehlen unbegrenzter Latenzverhalten. Sie zeigen lediglich, dass kritische Zustände unter den getesteten Bedingungen bislang nicht eingetreten sind.

Deterministische Systeme unterscheiden sich grundlegend dadurch, dass ihr zeitliches Verhalten durch architektonische Einschränkungen begrenzt wird und nicht aus statistischer Beobachtung abgeleitet wird.

Architektonische Konsequenzen

Die zuvor beschriebenen Missverständnisse beruhen auf einer gemeinsamen Annahme: dass deterministisches Verhalten aus Beobachtungen abgeleitet werden könne, anstatt es durch architektonische Kontrolle zu erzwingen. Diese Annahme mag für leistungsorientierte Systeme ausreichend sein, nicht jedoch für Systeme, in denen das Verpassen einer Deadline einen Fehler darstellt.

In allgemeinen Betriebssystemen wie Windows wird das Ausführungs-Timing von Kernel-Mechanismen beeinflusst, darunter Interrupt-Dispatching, Deferred Procedure Calls, Scheduler-Entscheidungen, Speicherverwaltungsaktivitäten und Treiberverhalten. Solange zeitkritische Ausführung diesen Mechanismen unterliegt, ohne vollständige Kontrolle über deren zeitliche Eigenschaften zu besitzen, können keine deterministischen Garantien gegeben werden.

Aus diesem Grund kann harte Echtzeitfähigkeit nicht allein durch erhöhte Prozesspriorität, Benchmarking oder Optimierung erreicht werden. Deterministisches Verhalten erfordert eine architektonische Trennung zwischen zeitkritischen und nichtkritischen Ausführungsdomänen. Zeitkritische Aufgaben müssen in einer Umgebung laufen, in der Interrupt-Behandlung, Scheduling-Verhalten und Ressourcenzugriffe kontrolliert und begrenzt sind, während nichtdeterministische Betriebssystemfunktionen unabhängig davon ausgeführt werden können.

Dieser Ansatz bewahrt die Ökosystemvorteile von Windows und ermöglicht gleichzeitig deterministisches Verhalten für zeitkritische Workloads. Die entscheidende Unterscheidung besteht darin, dass Determinismus nicht aus günstigem statistischem Verhalten innerhalb von Windows selbst entsteht, sondern aus architektonischer Kontrolle über die Ausführungsumgebung.

Was harte Echtzeit tatsächlich bedeutet

Ein System ist hart echtzeitfähig, wenn das Überschreiten einer definierten Zeitgrenze einen Systemfehler darstellt. Korrektheit hängt daher nicht nur von Rechenergebnissen ab, sondern auch von garantiertem zeitlichem Verhalten.

Um diese Anforderung zu erfüllen, müssen alle zeitrelevanten Ausführungspfade unter sämtlichen zulässigen Betriebsbedingungen begrenzt bleiben. Dazu gehören nicht nur der Anwendungscode selbst, sondern auch Scheduling-Entscheidungen, Interrupt-Behandlung, Speicherzugriffsverhalten sowie Interaktionen mit Hardware und Treibern. Wenn einer dieser Mechanismen unbegrenzte Verzögerungen verursachen kann, können keine deterministischen Garantien gegeben werden.

Harte Echtzeitsysteme benötigen daher mehr als nur geringe Latenz oder hohe Rechenleistung. Sie erfordern deterministisches Verhalten, explizite obere Zeitgrenzen und definierte Reaktionen auf Deadline-Verletzungen. Statistische Beobachtungen wie durchschnittliche Latenz, Jitter-Messungen oder lange stabile Laufzeiten können gute Leistung anzeigen, ersetzen jedoch keine garantierte Begrenzung der Ausführungszeit.

Harte Echtzeit wird daher nicht dadurch definiert, wie schnell ein System unter normalen Bedingungen arbeitet, sondern dadurch, ob sein Zeitverhalten unter allen relevanten Bedingungen kontrolliert bleibt.

Fazit

Harte Echtzeit beginnt dort, wo statistisches Denken endet. Systeme, die sich auf Messungen stützen, können Leistung nachweisen, aber keine Garantien liefern. Deterministisches Verhalten erfordert architektonische Kontrolle über alle zeitrelevanten Mechanismen. Ein System, das seinen Worst Case misst, kennt ihn nicht – es nimmt lediglich an, ihn bereits gesehen zu haben.

Imprint
Kithara Software GmbH, Alte Jakobstr. 78, 10179 Berlin, Germany, Phone: +49 30 2789673-0, Fax: +49 30 2789673-20
Email: info@kithara.com (general inquiries), support@kithara.com (technical inquiries), Internet: www.kithara.com
Copyright © 2026 Kithara Software GmbH. All rights reserved. No liability for printing errors. Subject to alterations.
Copying or reproducing, in whole or in part, is permitted only with approval of Kithara.
All named trademarks or registered trademarks are the properties of their respective owners. Mai, 2026