Zeit

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Systemsoftware
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2024/2025

Zeit

Soll ein Rechner korrekt simuliert werden, sind viele Zeitbedingungen korrekt zu simulieren:

- Anzahl der ausgeführten CPU-Instruktionen pro Zeiteinheit
- Anzahl der ausgeführten Grafik-Operationen pro Zeiteinheit
- Dauer einer Platten-Schreib- bzw. -Lese-Operation
- Dauer einer Netzwerk-Sende- bzw. -Empfangs-Operation
- Ticken des Uhrenbaustein
- Ticken des Timer-Baustein
- ...

Zeit

„Anzahl der ausgeführten CPU-Instruktionen pro Zeiteinheit" ist keine Konstante:

- verschiedene Instruktionen brauchen u.U. unterschiedlich lang
- Caching (Größe, Strategien, ...)
- Pipelining (Hazards)
- Sprungvorhersage (Größe Cache, Vorhersage-Strategie, ...)
- Super-Skalar-CPU (Anz. Execution-Einheiten, ...)
- variabler Takt (z.B. Taktreduktion bei Hitze)
- Multi-Threading (mit gegenseitigem Behindern)
- Multi-Core (mit gegenseitigem Behindern)
- nebenläufiges DMA
- ...

Zeit

Problem: Infos dazu stehen in keinem Handbuch.

„Lösung”: Wenn die Infos in keinem Handbuch stehen,

- ist jede Art von CPU-Zeitverhalten „korrekt”,
- müssen sich
  - Betriebssysteme und
  - Applikationen

auf unterschiedliche Geschwindigkeiten einstellen können
=> Timeouts sind i.A. schlechter Programmierstil!

Zeit

Zeitverhalten der I/O-Komponenten ähnlich unspezifiziert.

=> (Fast) jede Art von I/O-Zeitverhalten ist „korrekt".

=> Betriebssysteme sind entsprechend programmiert.

Zeit

Das Zeitverhalten der Uhren- und Timer-Bausteine ist i.A. genau beschrieben und einfach.

=> Genaue Simulation im Prinzip möglich.

Aber: Simulation dauert u.U. zu lang.

Zeit

Zwei Aspekte der „Zeit”:

Events in der virtuellen Maschine sollen

- in der korrekten Reihenfolge auftreten (kausale Reihenfolge)
  (das reicht für laufende Applikationen meist aus),
- in korrekten Zeitabständen auftreten
  (das ist für den realen User, für die reale Umgebung wichtig).

Zeit – Problembeispiel

Praktisch alle Komponenten reagieren in der Realität nicht „sofort”, wenn die CPU sie mit `in`- oder `out`-Instruktionen aufruft.

Zeit – Problembeispiel

```
void
disk_out(uint16_t port, uint8_t val)
{
  switch (port) {
  case 0: buf = val; break;
  case 1: pos = val; break;
  case 2:
    if (pos < SIZE) {
      if (val) disk[pos] = buf;
      else buf = disk[pos];
      status = OK;          <--- Problematisch!
      interrupt();          <--- Problematisch!
    } else {
      status = BAD;
    }
    break;
  }
}
```

Zeit – Problembeispiel

```
void
disk_out(uint16_t port, uint8_t val)
{
  ...
      else buf = disk[pos];
      status = BUSY;
      delay = 1000;
    } else {
      status = BAD;
  ...
}

void
disk_step()
{
  if (delay)
    if (--delay == 0) {
      status = OK;
      interrupt();
    }
}
```

Zeit – Problem

VM z.T. langsamer, z.T. schneller als reale Maschine.

- VM z.T. zu langsam:
  - Simulation der VM aufwändig
  - nebenläufige Prozesse auf Host-OS
  - ...
- VM z.T. zu schnell:
  - im Speicher simulierte Platten schneller als echte Hardware
  - auf einem realen Host simulierte Rechner kommunizieren lokal schneller als über's Netz
  - Power-off- oder Halt-Zustand sehr einfach simulierbar
  - ...

Zeit – Problem

Läuft die VM zu langsam, können Timeouts auftreten, die in der Realität nicht aufgetreten wären.

Linux, Windows, usw. garantieren für ihre System-Calls keine Zeitbedingungen. Aber Linux-, Windows-, usw. -Betriebssysteme kennen intern Timeouts für

- Hardware,
- Netzwerk-Protokolle.

Werden diese Timeouts verletzt, werden Fehlermeldungen produziert („Platte reagiert nicht", „Server down" u.ä.).

Zeit – Problem

Moderne Desktop-Betriebssysteme garantieren i.A. keine Antwortzeiten. Moderne CPUs sind vom Timing her extrem schwer einzuschätzen. Trotzdem benutzen immer wieder Programmierer Timeouts.

Dies kann auf virtuellen Rechnern (und langsamen realen Rechnern!) zu Problemen führen.

Beispiel GNOME:

GNOME startet beim Einloggen seine Applets. Melden sich diese nach dem Start nicht rechtzeitig als „laufend", fragt GNOME ob es das Applet für die Zukunft disablen soll.

$=>$ Timeouts sind generell von Applikation-Programmierern zu vermeiden!

Zeit – Problem

Läuft die VM zu schnell können ebenfalls Probleme auftreten, die in der Realität nicht aufgetreten wären.

Beispiel aus Linux-2.5:

OS schickt *erst* ein Kommando an das Keyboard und setzt *dann* den Interrupt-Handler auf.

In der Realität funktioniert dies, da die Kommunikation mit dem Keyboard über eine langsame serielle Leitung geschieht und die CPU damit genügend Zeit hat, ihren Interrupt-Handler aufzusetzen.

Kommt jedoch die Antwort des Keyboards und damit der Interrupt zu früh, geht die Antwort verloren (=> „No keyboard detected.").

Zeit – Problem

Denkbare Lösungen:

1. virtuelle Zeit läuft gemäß dem Simulationsfortschritt in der VM
2. virtuelle Zeit entspricht der realen Zeit
3. Kombination aus 1 und 2

Zeit – Virtuelle Zeit gemäß realer Zeit

„Virtuelle Zeit gemäß realer Zeit”:

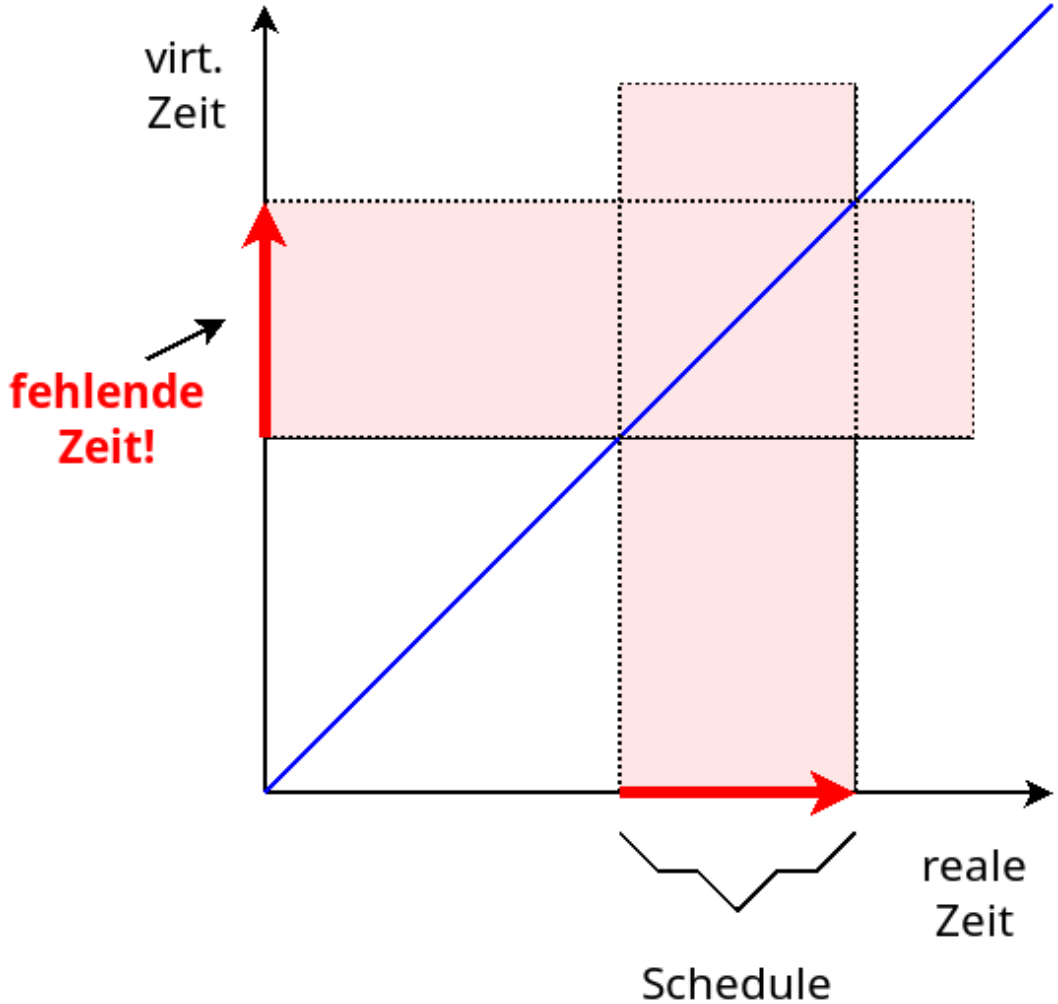
Nachteile:

- Interaktion der Komponenten untereinander schwierig (einzelne Komponenten u.U. zu langsam (Scheduling des Hosts!) oder zu schnell).
- Real-Zeit-Bedingungen i.A. nicht einzuhalten.

Vorteile:

- Interaktion mit der realen Umwelt (User, Internet, ...) u.U. möglich.
- Mischen von VMs und realen Maschinen möglich.

Zeit – Virtuelle Zeit gemäß realer Zeit



Zeit – Virtuelle Zeit gemäß Simulationsfortschritt

„Virtuelle Zeit gemäß Simulationsfortschritt”:

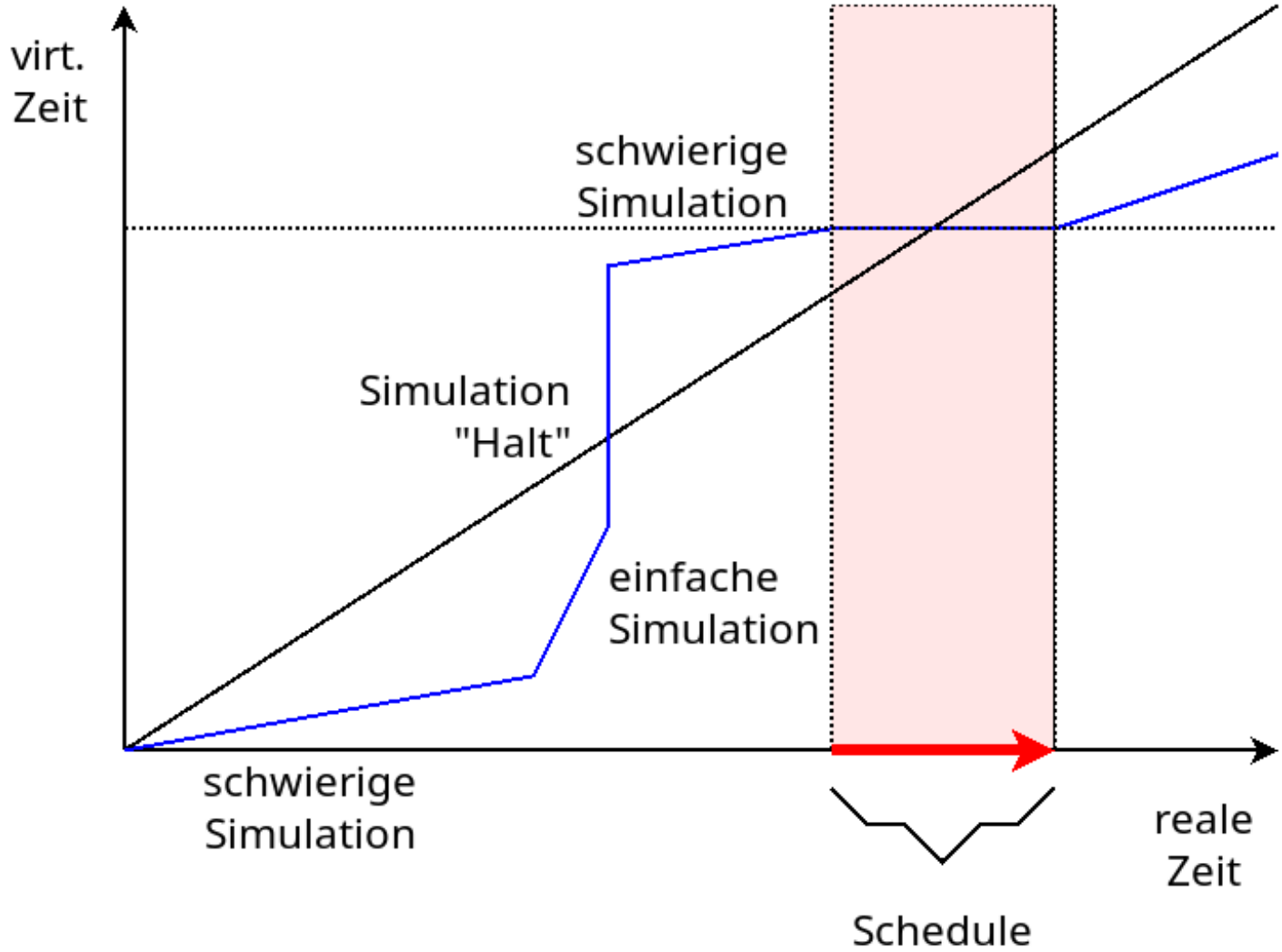
Nachteile:

- Interaktion mit der realen Umwelt (User, Internet, ...) unmöglich (Timeouts).
- Reale Umwelt muss durch virtuelle Umwelt (virt. User, virt. Internet, ...) ersetzt werden (Aufwand).

Vorteile:

- VM kann deterministisch laufen (Reproduzierbarkeit).
- Zeitbedingungen in Hard- und Software sind kein Problem (Exaktheit).
- Simulation kann angehalten/fortgesetzt werden (Debugging).

Zeit – Virtuelle Zeit gemäß Simulationsfortschritt



Zeit – Dynamische Virtuelle Zeit

„Dynamische Zeit”:

Idee:

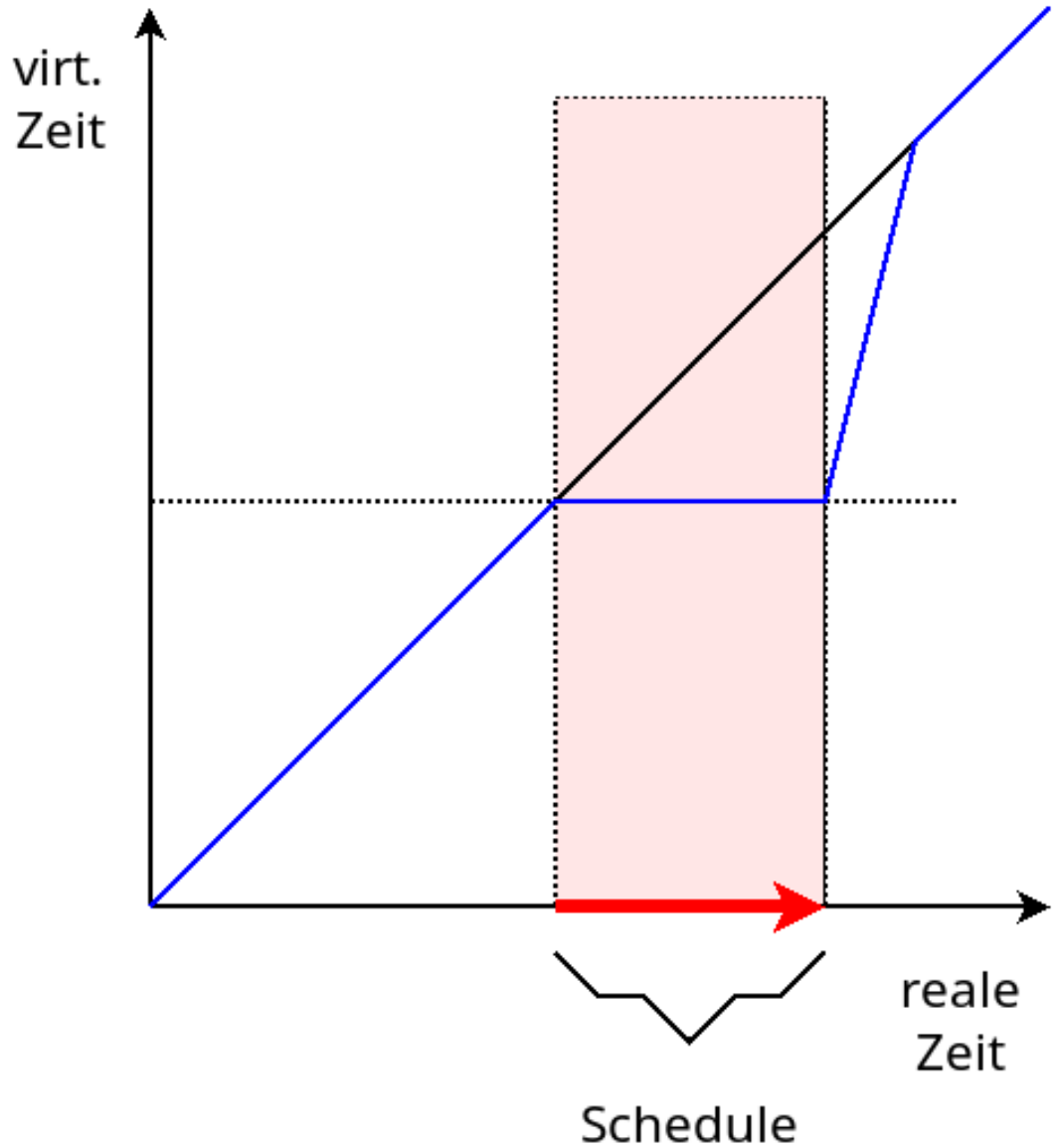
Soweit möglich soll virtuelle Zeit der realen Zeit entsprechen.

Aber:

Die Hardware soll aber pro realer Zeiteinheit eine Mindestanzahl von virtuellen Zeiteinheiten simulieren.

Ggf. wird die virtuelle Zeit verlangsamt. Wenn möglich wird die verlorene Zeit später wieder aufgeholt.

Zeit – Dynamische Virtuelle Zeit



Zeit – Dynamische Virtuelle Zeit

Nachteile:

- Zeitverhalten schwer abzuschätzen.
- Zeitverhalten abhängig von Load des Hosts.

Vorteile:

- Interaktion mit der realen Umwelt (User, Internet, ...) u.U. möglich.
- Mischen von VMs und realen Maschinen möglich.

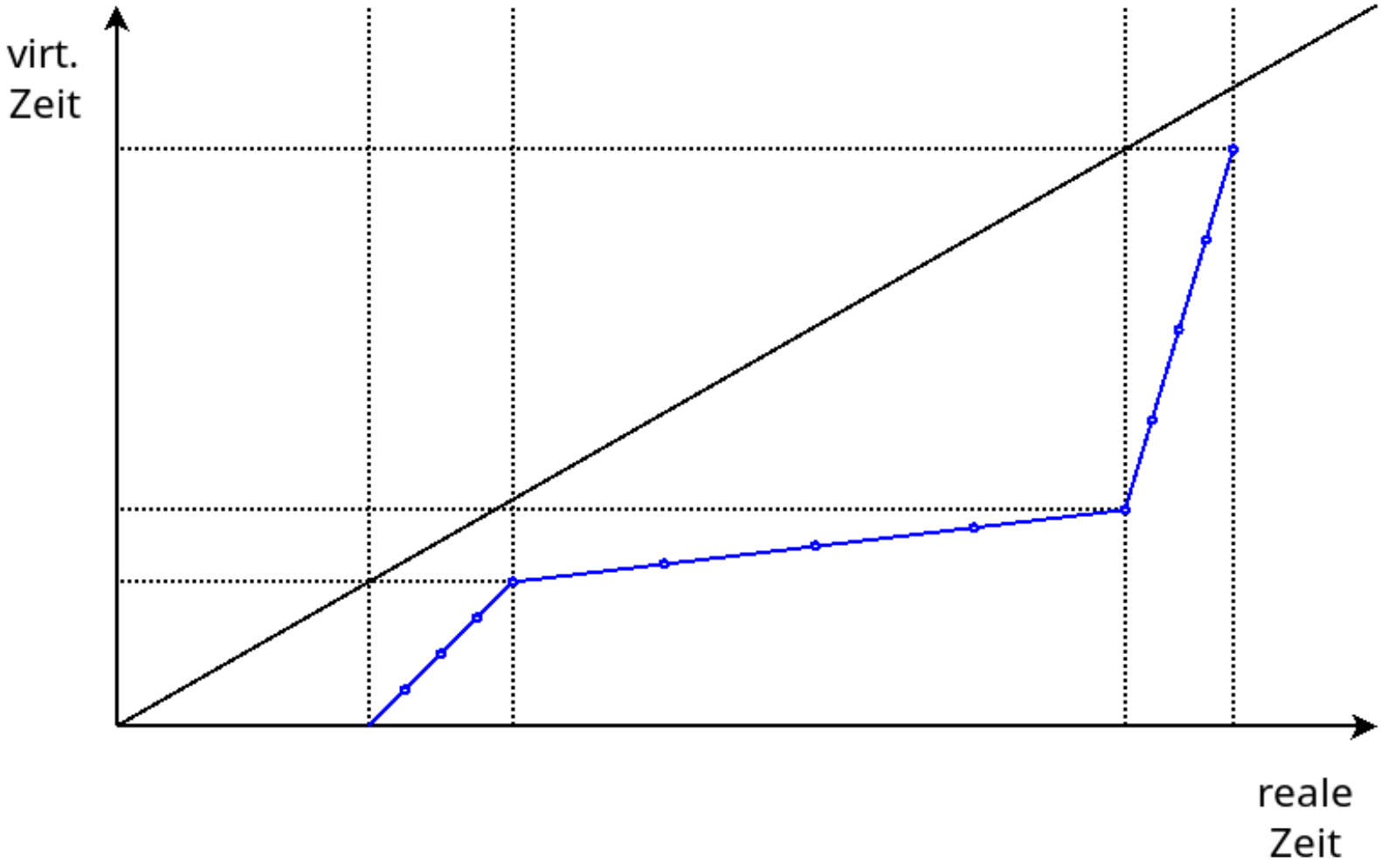
Zeit – Real-Zeit-Ermittlung

Um die virtuelle Zeit der Real-Zeit anzupassen, muss die reale Zeit häufig ermittelt werden.

Uhrzeit-Abfrage unter Windows/Linux (System-Call) sehr zeitaufwändig...

=> Abfrage darf nur selten geschehen. Zeit zwischen gemessenen Zeiten interpolieren.

Zeit – Dynamische Virtuelle Zeit



Zeit – Dynamische Virtuelle Zeit

Algorithmus: Update der virtuellen Zeit nach jeder ausgeführten Instruktion:

```
void
update_time()
{
  icnt++;
  if (icnt == COUNT) {
    r0 = r1;
    r1 = gettime();
    icnt = 0;
  }
}

unsigned int
virt_time()
{
  return r0 + icnt * (r1 - r0) / COUNT;
}
```

Zeit – Dynamische Virtuelle Zeit

Vorteile des Algorithmus:

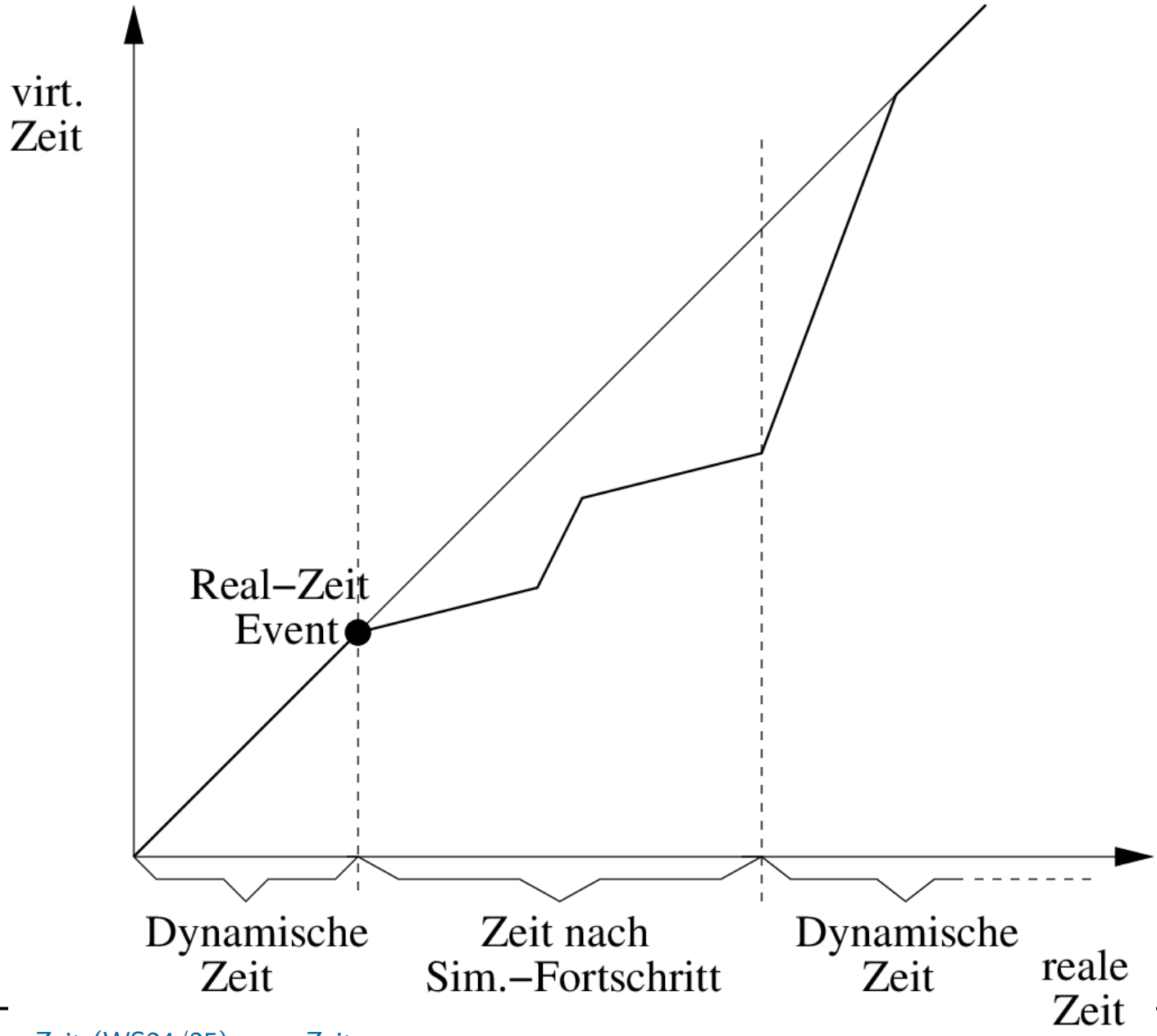
- `update_time()` ist sehr schnell ausführbar.
- `gettime()` wird selten aufgerufen.
- `virt_time()` ist schnell ausführbar (mit `COUNT` Zweierpotenz).
- Ein Scheduling des Host hat keinen (großen) Einfluss auf die Zeit. Die virtuelle Zeit schreitet nur voran, wenn Instruktionen simuliert werden.

Zeit – Zeitweise deterministische Zeit

- Manchmal *muss* die virtuelle Zeit sehr gut mit dem Simulationsfortschritt übereinstimmen. Z.B. wenn die CPU auf Real-Zeit-abhängige Komponenten zugreift.
- Manchmal *muss* die virtuelle Zeit gut mit der realen Zeit übereinstimmen. Z.B. wenn ein realer User mit der virtuellen Maschine kommunizieren will.

=> Umschalten zwischen „Dynamische Zeit" und „Zeit gemäß Simulationsfortschritt"

Zeit – Zeitweise deterministische Zeit



Zeit – Zeitweise deterministische Zeit

- Wechsel „Dyn. Zeit” $\rightarrow$ „Zeit gemäß Sim.-Fortschritt”, wenn besondere Events auftreten (z.B. Zugriff auf Uhr oder I/O-Gerät)
- Rückwechsel, wenn eine bestimmte Zeit kein besonderer Event mehr aufgetreten ist

Heuristik:

- Was gilt als „besonderer Event”?
- Nach welcher Zeit erfolgt Rückwechsel?