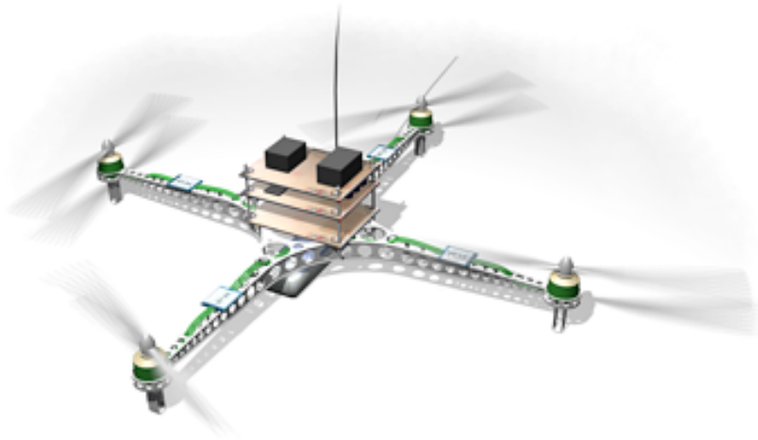
Cooperative Memory Management in Safety-Critical Embedded Systems

Isabella Stilkerich

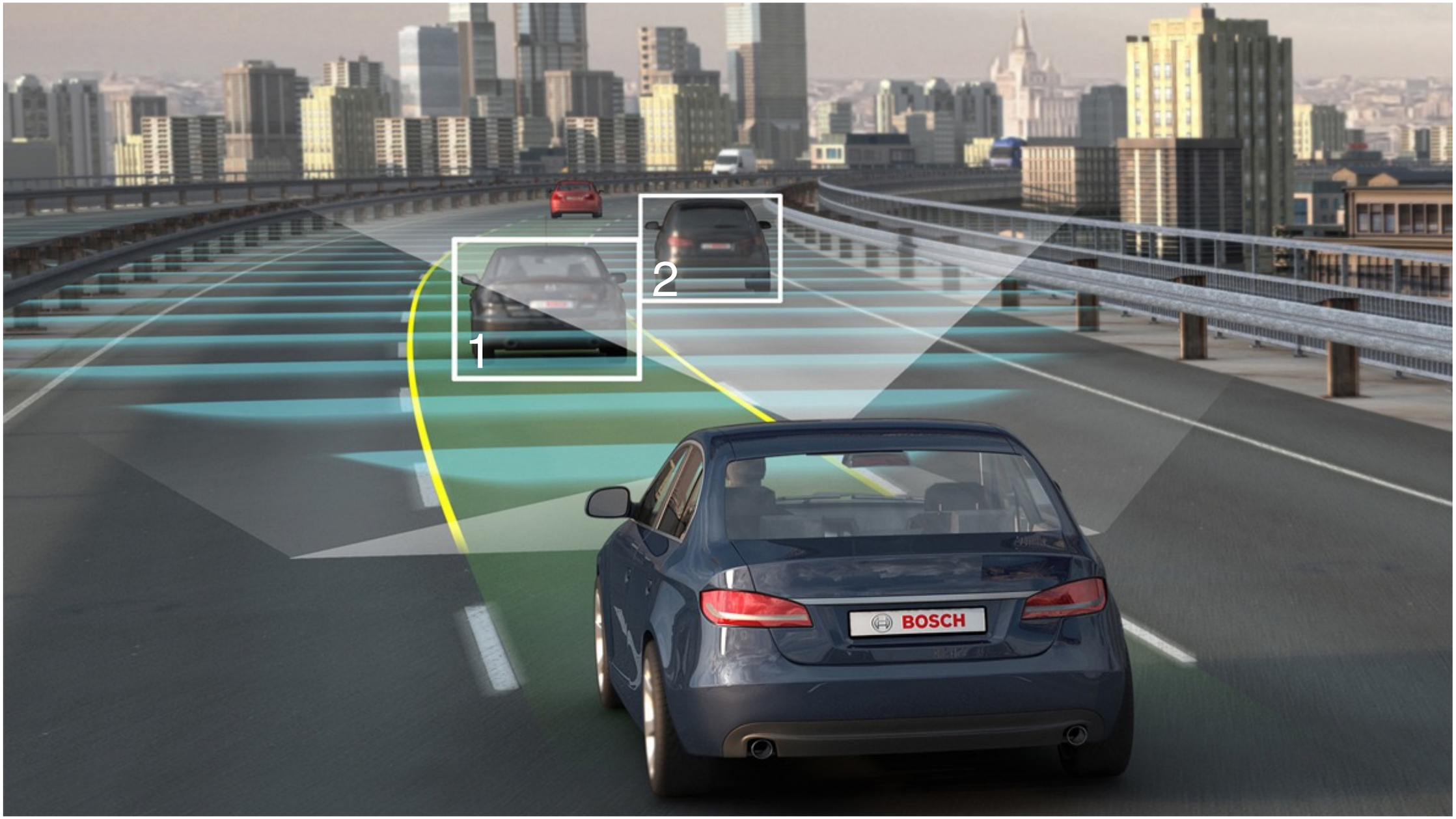
Department of Computer Science 4
Distributed Systems and Operating Systems

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Motivation



Beispiel: Kollisionserkennung



Speicherverwaltung

- Programme brauchen Speicherressourcen, um ihre Aufgabe umzusetzen
- Verwaltung physikalischen Speichers
  - Bereitstellung von Speicherplatz für Anwendungsdaten

Freier Speicher:

Belegter Speicher:

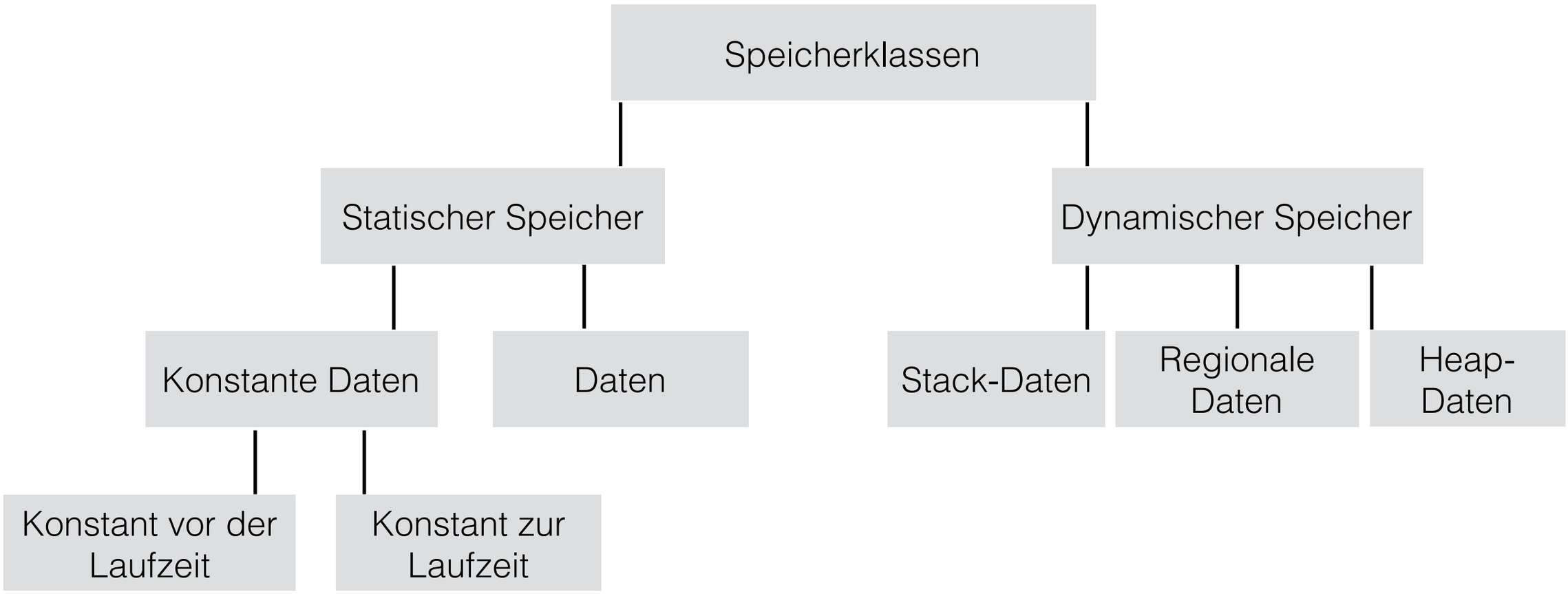
Speicherverwaltung

- Speicherverwaltung ist keine triviale Aufgabe
  - Adäquate Platzierung: manuell, (teil-)automatisiert
  - Verwaltungstechnik: manuell, (teil-)automatisiert
  - Berücksichtigung des Speichernutzungsverhaltens der Anwendung

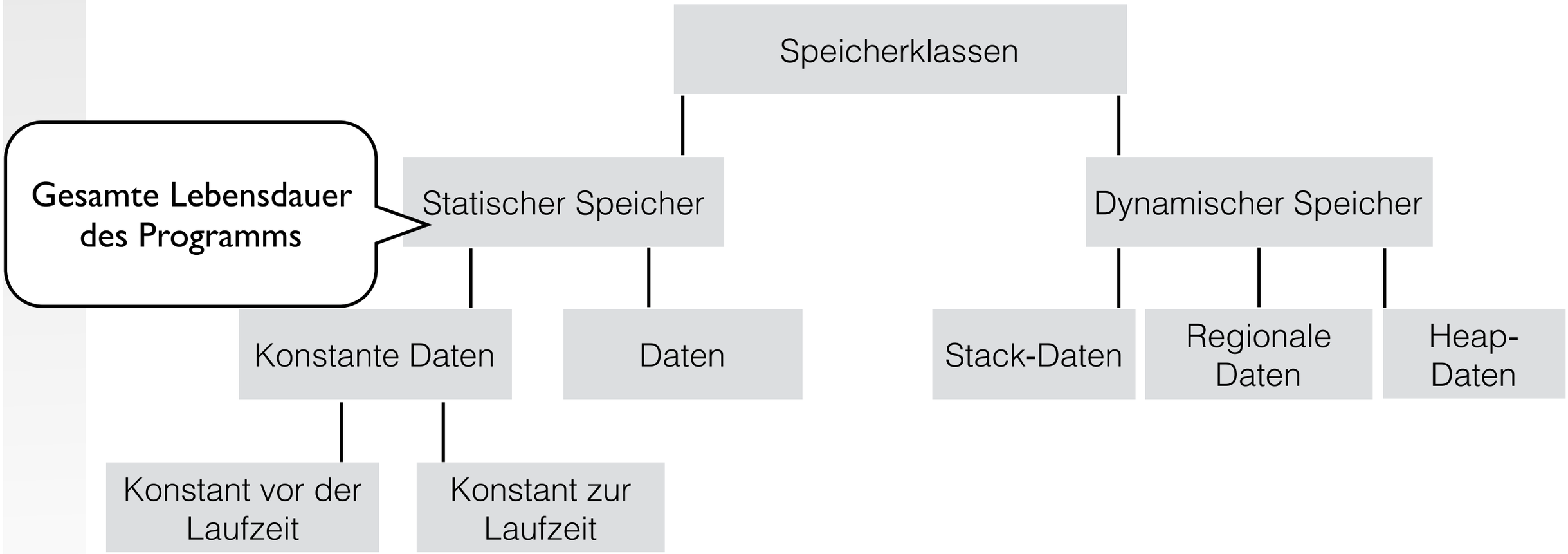
Freier Speicher:

Belegter Speicher:

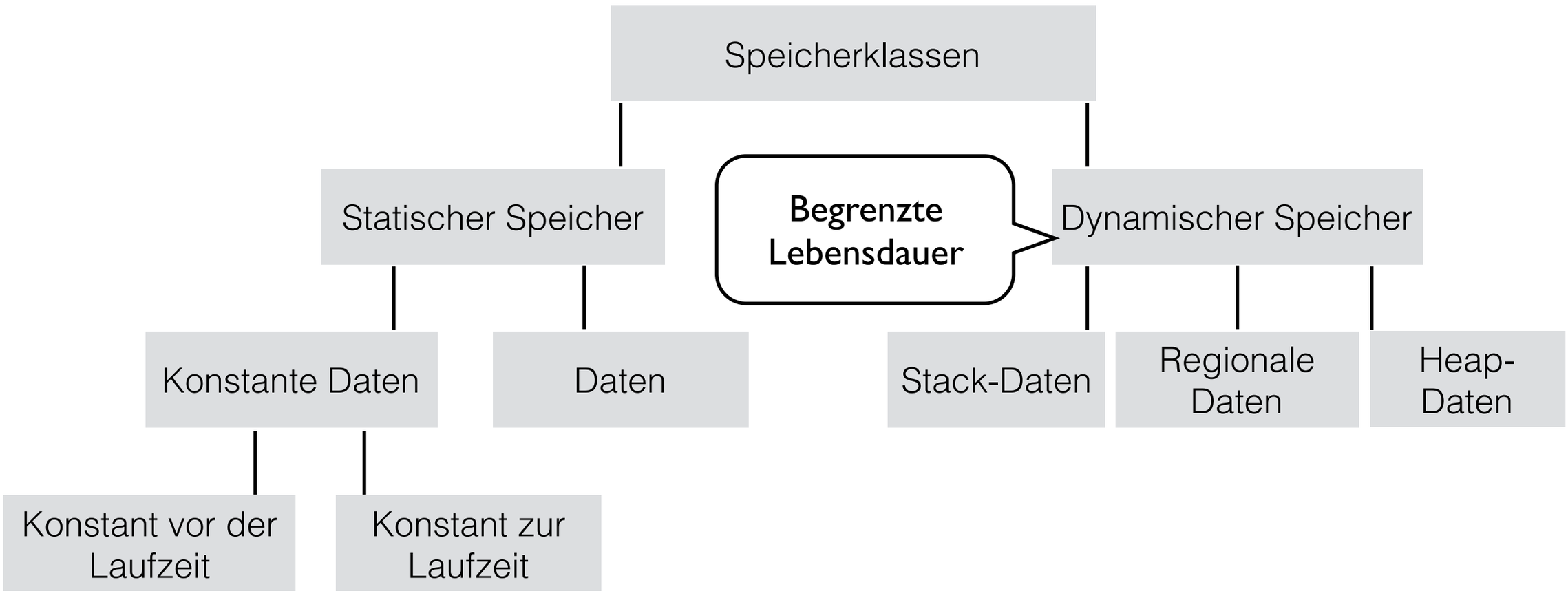
Hybrides Speichermodell



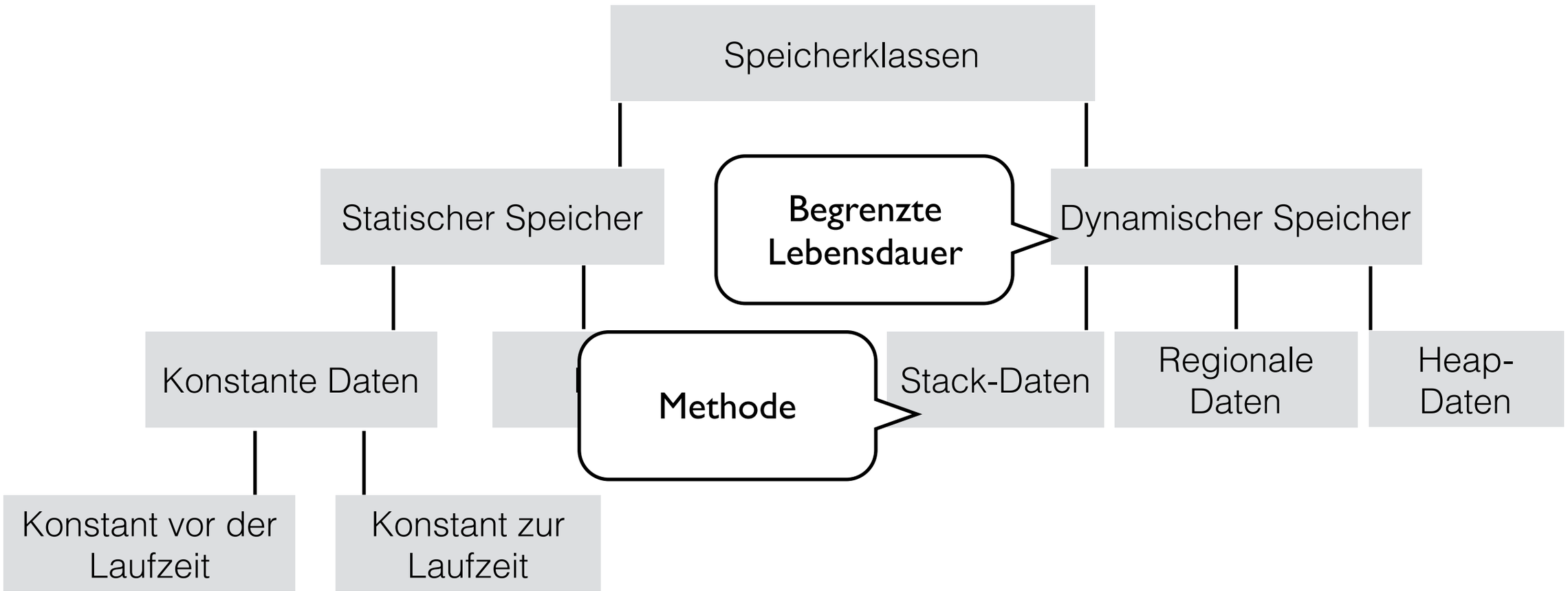
Hybrides Speichermodell



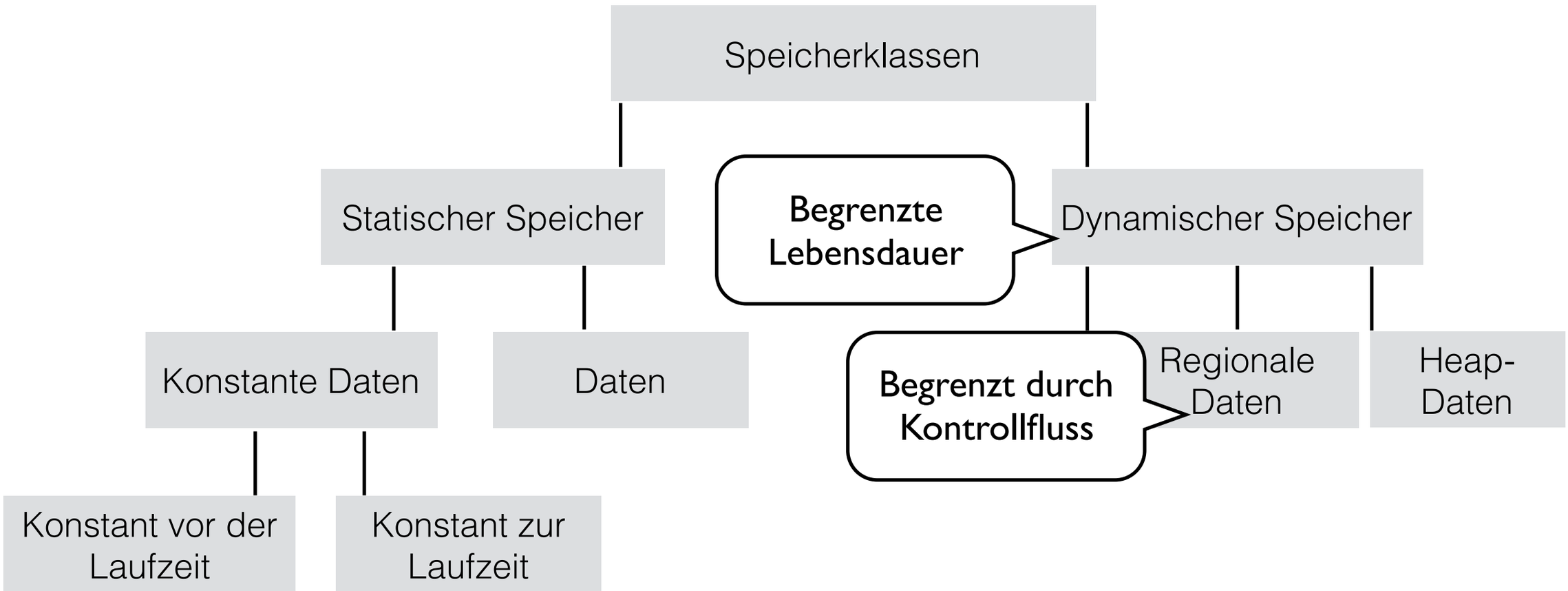
Hybrides Speichermodell



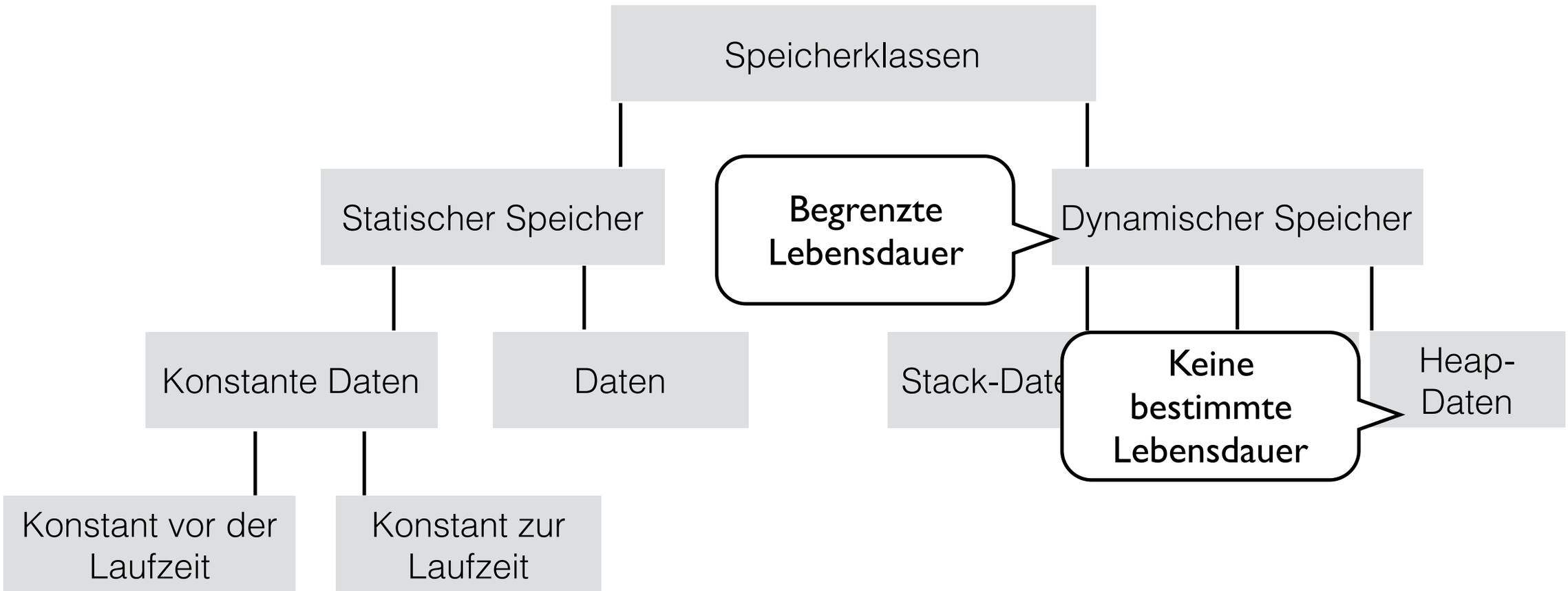
Hybrides Speichermodell



Hybrides Speichermodell

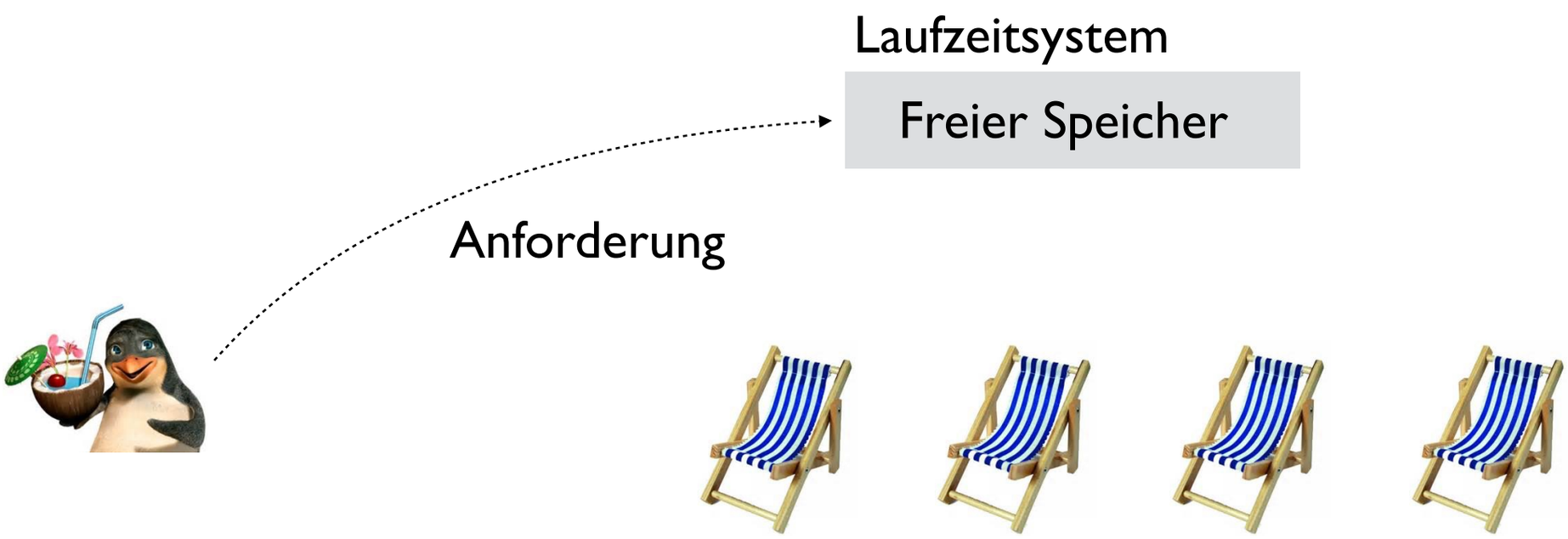


Hybrides Speichermodell



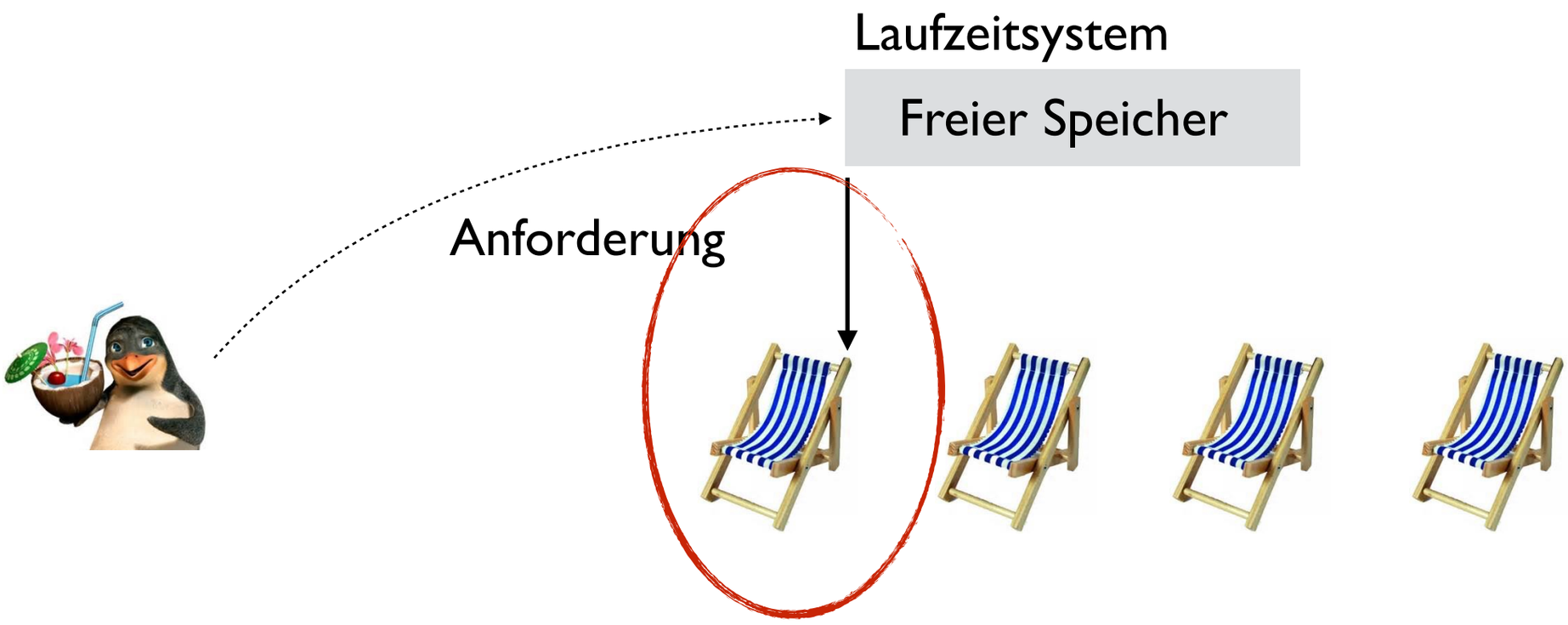
Automatisierte Speicherverwaltung

- Automatisierung als Ansatz, Komplexität zu begegnen
- Programmiersprachen mit Laufzeitsystem als Basis
  - Auswahl eines geeigneten Speicherplatzes
  - Freigabe des Speichers, falls dieser nicht mehr referenziert wird
  - Verlässliches Erkennen von Referenzen fördert Automatisierung



Automatisierte Speicherverwaltung

- Automatisierung als Ansatz, Komplexität zu begegnen
- Programmiersprachen mit Laufzeitsystem als Basis
  - Auswahl eines geeigneten Speicherplatzes
  - Freigabe des Speichers, falls dieser nicht mehr referenziert wird
  - Verlässliches Erkennen von Referenzen fördert Automatisierung

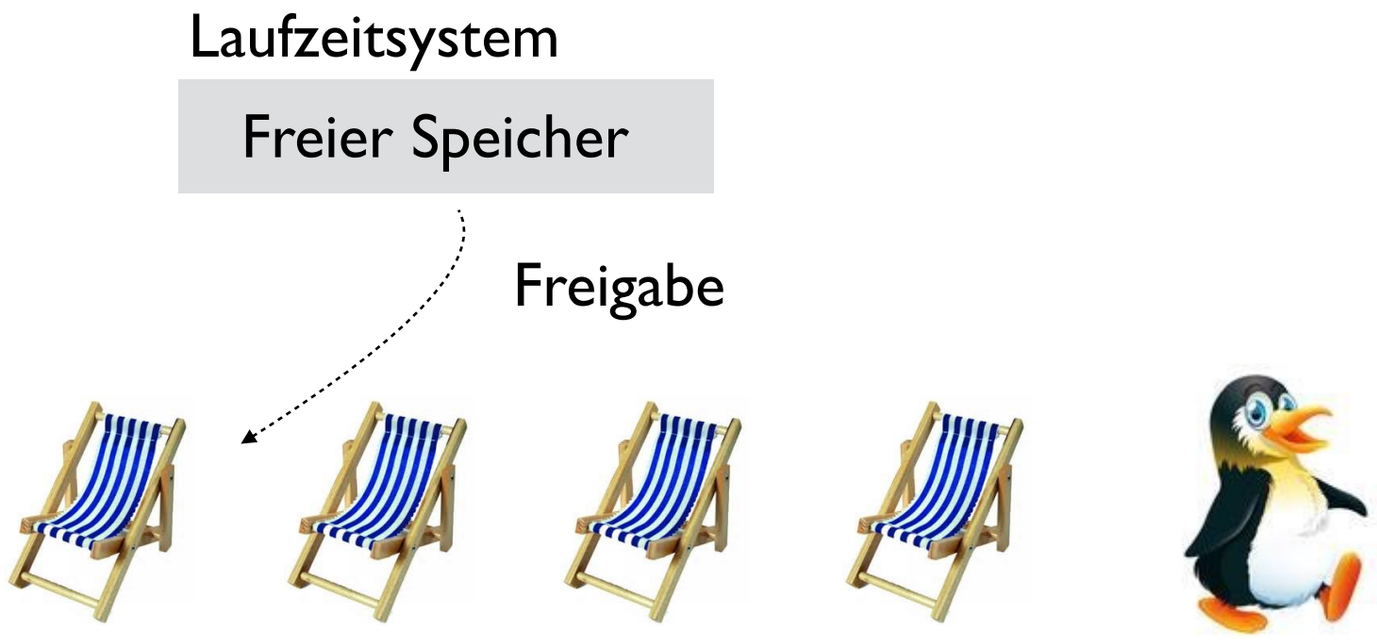


Automatisierte Speicherverwaltung

- Automatisierung als Ansatz, Komplexität zu begegnen
- Programmiersprachen mit Laufzeitsystem als Basis
  - Auswahl eines geeigneten Speicherplatzes
  - Freigabe des Speichers, falls dieser nicht mehr referenziert wird
  - Verlässliches Erkennen von Referenzen fördert Automatisierung

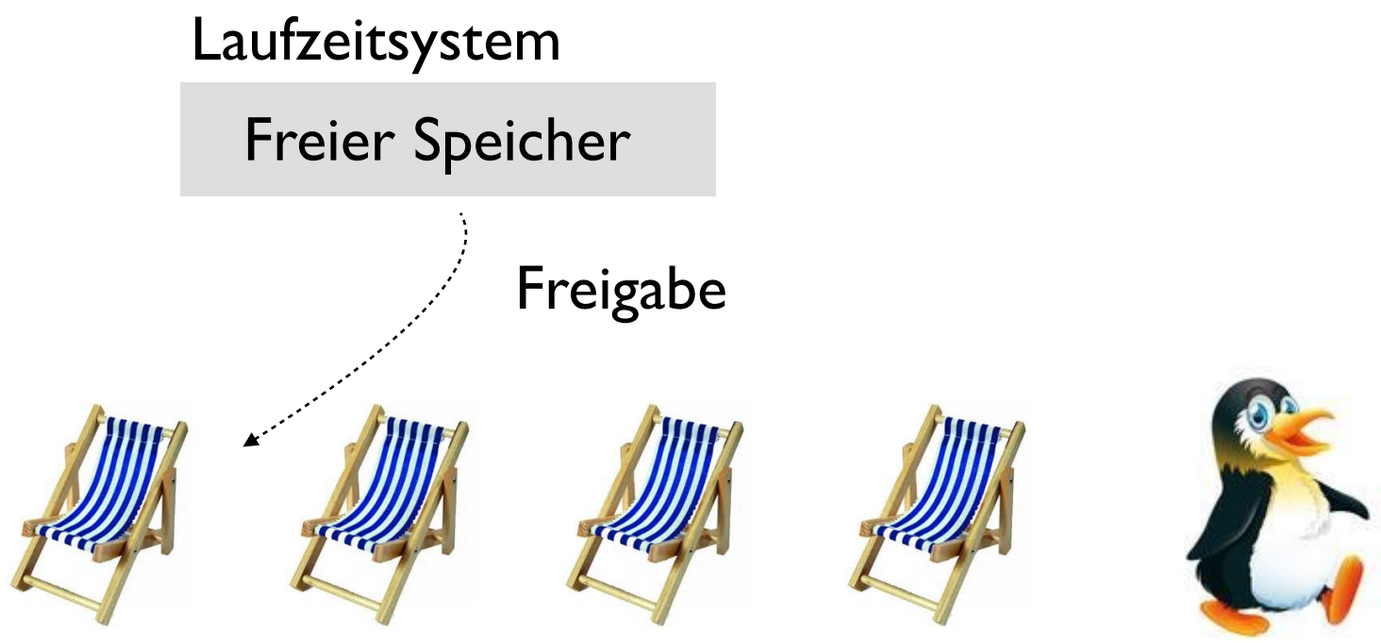
Automatisierte Speicherverwaltung

- Automatisierung als Ansatz, Komplexität zu begegnen
- Programmiersprachen mit Laufzeitsystem als Basis
  - Auswahl eines geeigneten Speicherplatzes
  - Freigabe des Speichers, falls dieser nicht mehr referenziert wird
  - Verlässliches Erkennen von Referenzen fördert Automatisierung



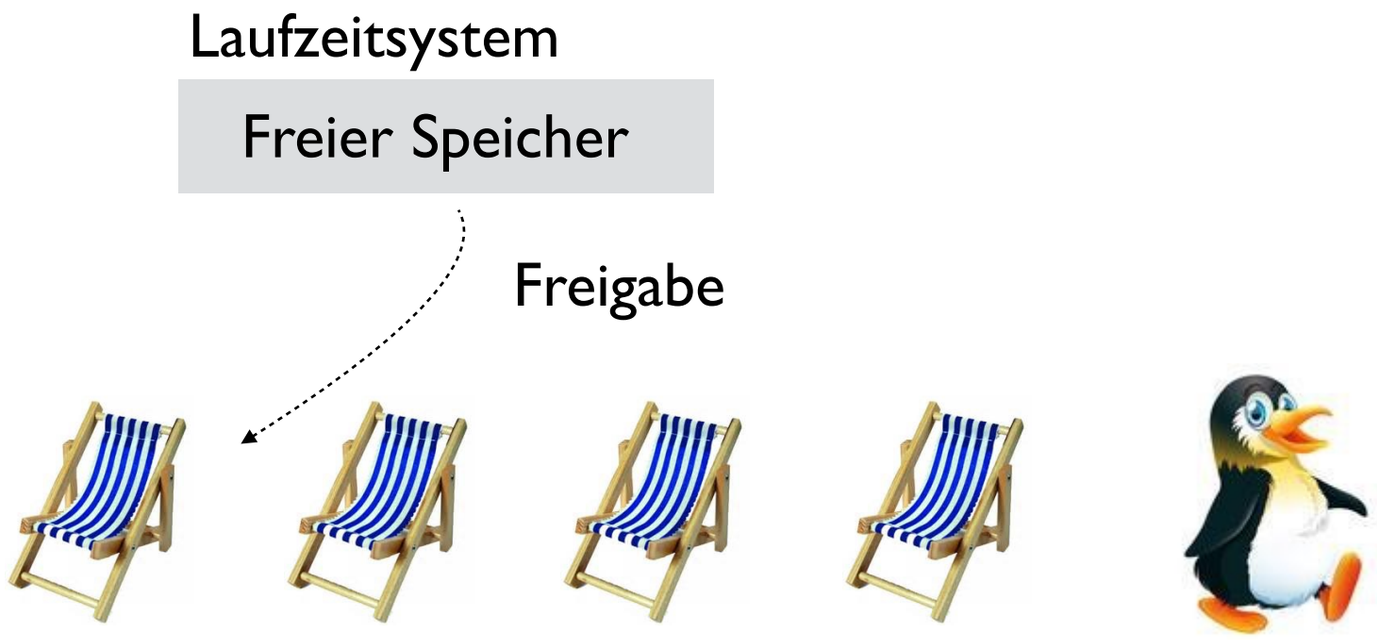
Automatisierte Speicherverwaltung

- Vertreter der verwalteten Sprachen: Java
- Typ- und speichersicher:
  - Zum Objekttyp definierte Operationen
  - Zeiger verweisen auf gültige Objekte
  - Zeiger auf freigegebenen Speicher existieren nicht



Automatisierte Speicherverwaltung

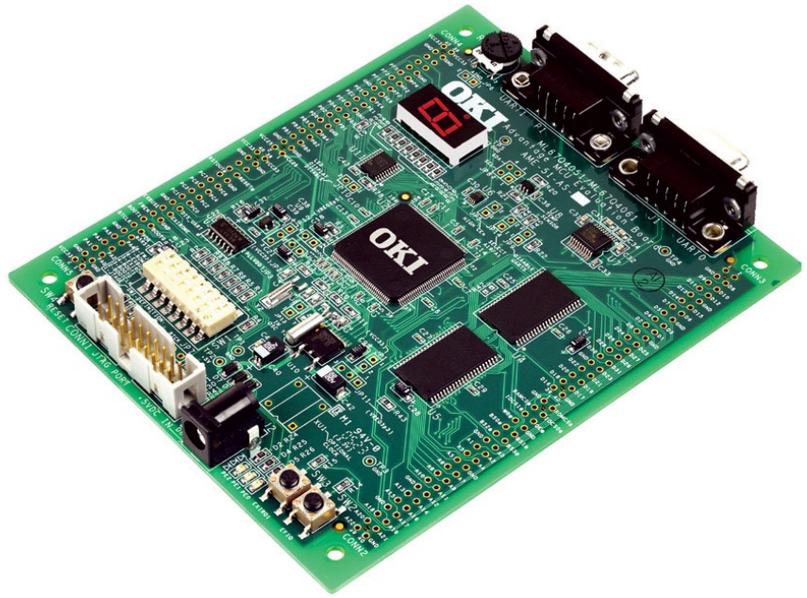
- Sicherstellung durch Compiler und/oder Laufzeitumgebung
  - Statische/dynamische Prüfungen
  - Implizite Speicherverwaltung
- Implizite Verfahren haben Vor- und Nachteile
  - Eignen sich für eine bestimmte Speicherklasse
  - Hybrider Ansatz ist sinnvoll



Speicherlandschaft

Transiente Hardware-Fehler

ECC-Schutz



Interne Speicher

Persistenz

Externe Speicher

Größe

Zugriffszeiten

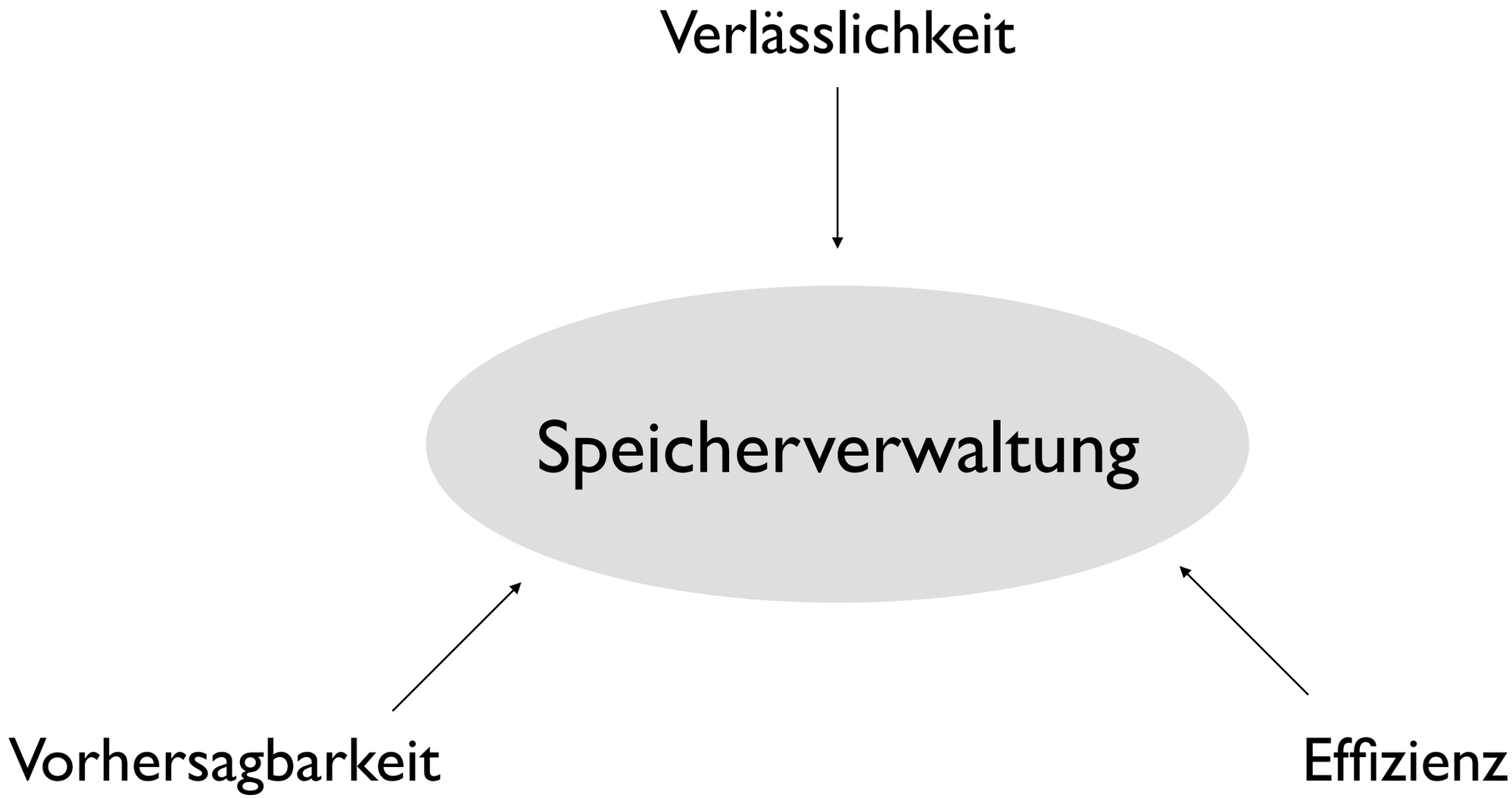
Forschungsfrage

Wie kann eine **maßgeschneiderte Speicherverwaltung** für **Anwendungsprogramm** und die verwendete **Hardware** bereitgestellt werden?

Entwurfskriterien

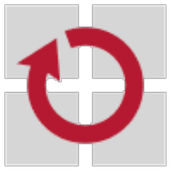
1. Berücksichtigung von Randbedingungen
2. Bedienung des anwendungsspezifischen Speichernutzungsverhaltens
   - Automatisierte Einordnung von Daten in Speicherklassen
   - Auswahl der Verwaltungstechnik

Co-Design: Randbedingungen



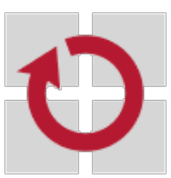
Randbedingungen

- **Verlässlichkeit** in Anwesenheit transienter Fehler?
  - Einfügen von Integritätsprüfungen
  - Vermeidung von Fehlern
- **Ressourceneffizienz** und **Vorhersagbarkeit**?
  - Anwendungsspezifisches Speichernutzungsverhalten
  - Automatisierte Bestimmung und Verwaltung von Speicherklassen



Die KESO Java Virtual Machine

- Implementierung der Speicherverwaltung innerhalb von KESO
- Vollständige Übersetzung von Java nach C vor der Laufzeit
- Statische Konfiguration
- Maßschneiderung der Laufzeitumgebung
- Generativer Ansatz



Kooperative Speicherverwaltung (CMM)

Eigenschaften der Hardware

Speicher

Layout des Speichers

Größe des Speichers

Rate transienter Fehler

…

Typsichere Anwendung

Klassenhierachie

Aufrufgraph

(De-)Allokations-verhalten

Kommunikation

…

Konfiguration der Anwendung

Rate der zu erkennenden transienten Fehler

Stack-Allokation

Thread-lokaler Speicher

Echtzeit Garbage Collector

…

Java Übersetzer

Escape-Analyse

Integritätsüberprüfungen

…

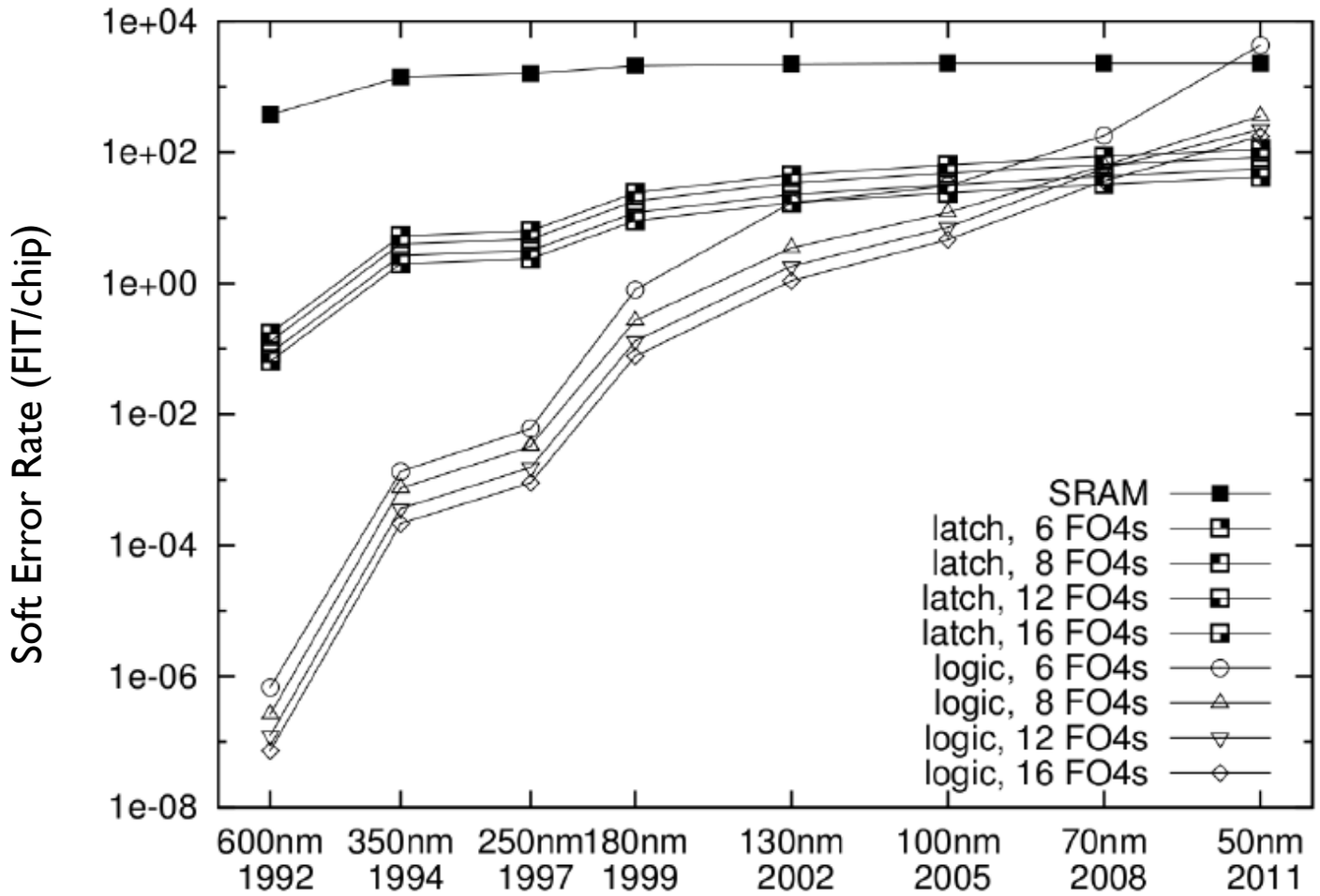
Anwendung inklusive Speicherverwaltung

Randbedingungen

- **Verlässlichkeit** in Anwesenheit transienter Fehler?
  - Einfügen von Integritätsprüfungen
  - Vermeidung von Fehlern
- **Ressourceneffizienz** und **Vorhersagbarkeit**?
  - Anwendungsspezifisches Speichernutzungsverhalten
  - Automatisierte Bestimmung von Speicherklassen

Typsicherheit und transiente Fehler

- Transiente Fehler werden wahrscheinlicher
- Strukturgrößenverkleinerung, Umwelteinflüsse
- Manifestieren sich durch temporäre Fehler (Bitkipper)



Bitkipper

Speicher 10110110 ⟶ 10111110

Typsicherheit und transiente Fehler

- Bitkipper können die Typsicherheitseigenschaft zerstören
- Maßnahmen nötig: Verlässliche Speicherverwaltung ermöglichen

Fehlerhypothese

1. Transiente Fehler sind an der Programmierschnittstelle sichtbar
2. Schutz des Laufzeitsystems, kein Anwendungsschutz [1]
3. Verlässliche Speicherbasis ist der Flashspeicher
   - Flashspeicher robuster gegen transiente Fehler (Cellere et al., IDEM 2008)
   - Programmspeicher und dort abgelegte Daten sind bereits geschützt
4. Kurzlebige Daten sind weniger fehleranfällig

JTRES '11 [1] Stilkerich et al: Automated Application of Fault-Tolerance Mechanisms in a Component-Based System

Umgang mit transienten Fehlern

- Behandlung von Fehlern zur Laufzeit
  - Integritätsprüfungen der kritischen Stellen des Laufzeitsystems
- Vermeidung von Fehlerauswirkungen
  - Verringerung der Angriffsfläche
  - Optimierungen im KESO Übersetzer

Umgang mit transienten Fehlern

- Behandlung von Fehlern zur Laufzeit
  - Integritätsprüfungen der kritischen Stellen des Laufzeitsystems
- Vermeidung von Fehlerauswirkungen
  - Verringerung der Angriffsfläche
  - Optimierungen im KESO Übersetzer

Kritische Stellen des Laufzeitsystems

Anwendung und Laufzeitsystem sind vermischt:

- Referenzen in der Anwendung [2]
  - Adressen
  - Verwaltungsinformationen in Objekten
- Speicherverwaltung [3]
  - Zugriff auf Referenzen der Anwendung
  - Zugriff auf interne Zeiger der Laufzeitumgebung
  - Zugriff auf interne Daten der Laufzeitumgebung

LCTES ’13 [2] Stilkerich et al: A JVM for Soft-Error-Prone Embedded Systems
CASES ’14 [3] Stilkerich et al: Team Up: Cooperative Memory Management in Embedded Systems

Kritische Stellen des Laufzeitsystems

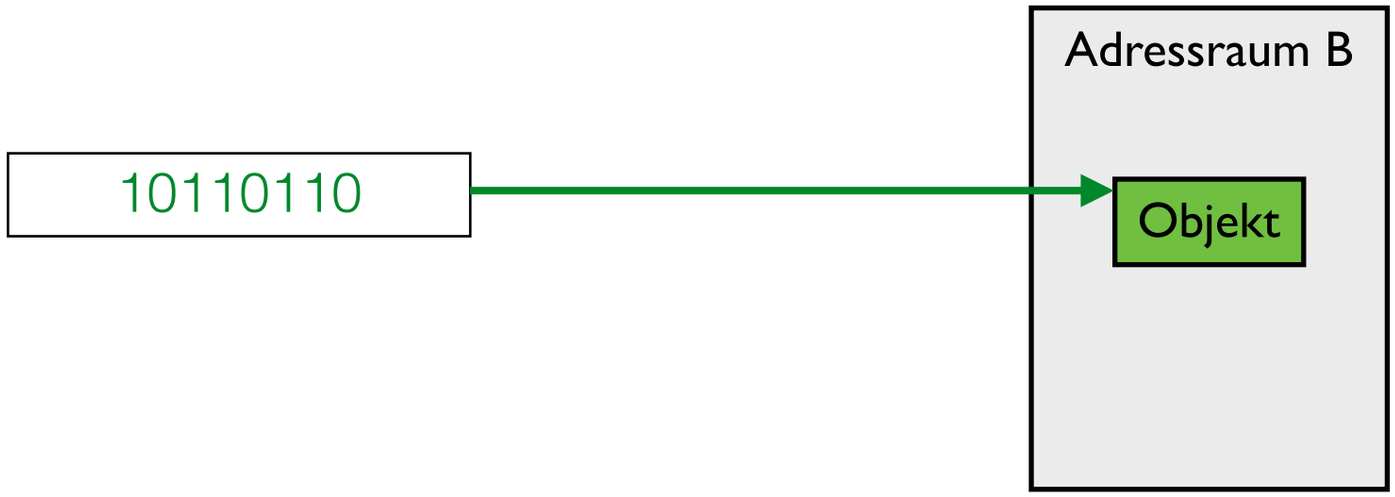
Anwendung und Laufzeitsystem sind vermischt:

- Referenzen in der Anwendung [2]
  - Adressen
  - Verwaltungsinformationen in Objekten
- Speicherverwaltung [3]
  - Zugriff auf Referenzen der Anwendung
  - Zugriff auf interne Zeiger der Laufzeitumgebung
  - Zugriff auf interne Daten der Laufzeitumgebung

LCTES '13 [2] Stilkerich et al: A JVM for Soft-Error-Prone Embedded Systems
CASES '14 [3] Stilkerich et al: Team Up: Cooperative Memory Management in Embedded Systems

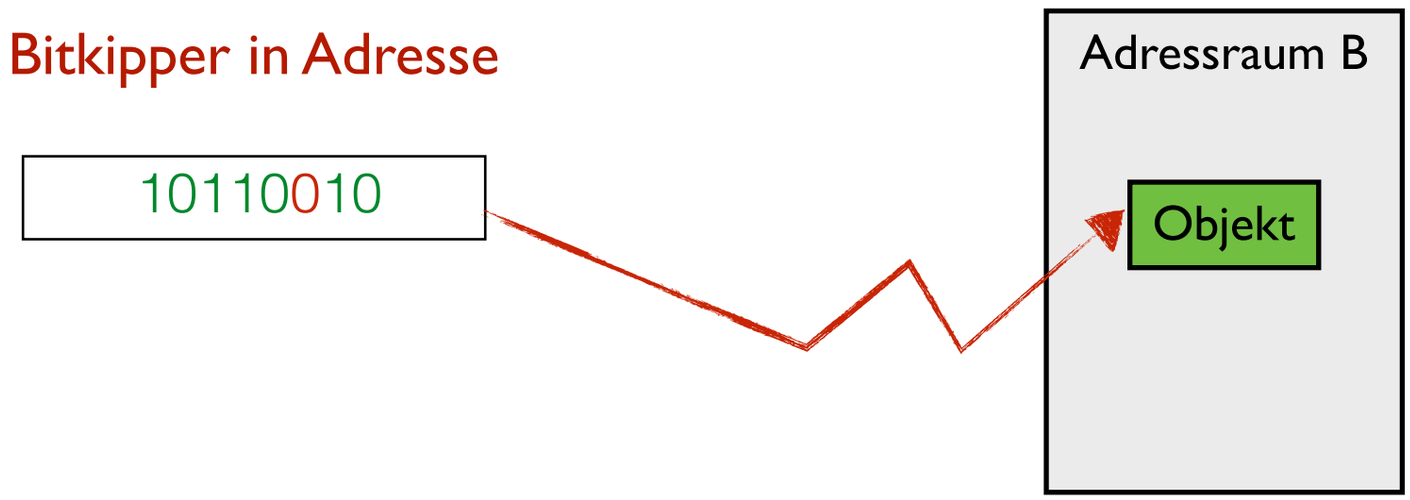
Wilde Referenzen

- Java-Referenz: Speicheradresse der Daten
- Kaputte Referenz: Wilder Zeiger
- Schlimmster Fall: Zugriff auf beliebigen Speicherbereich



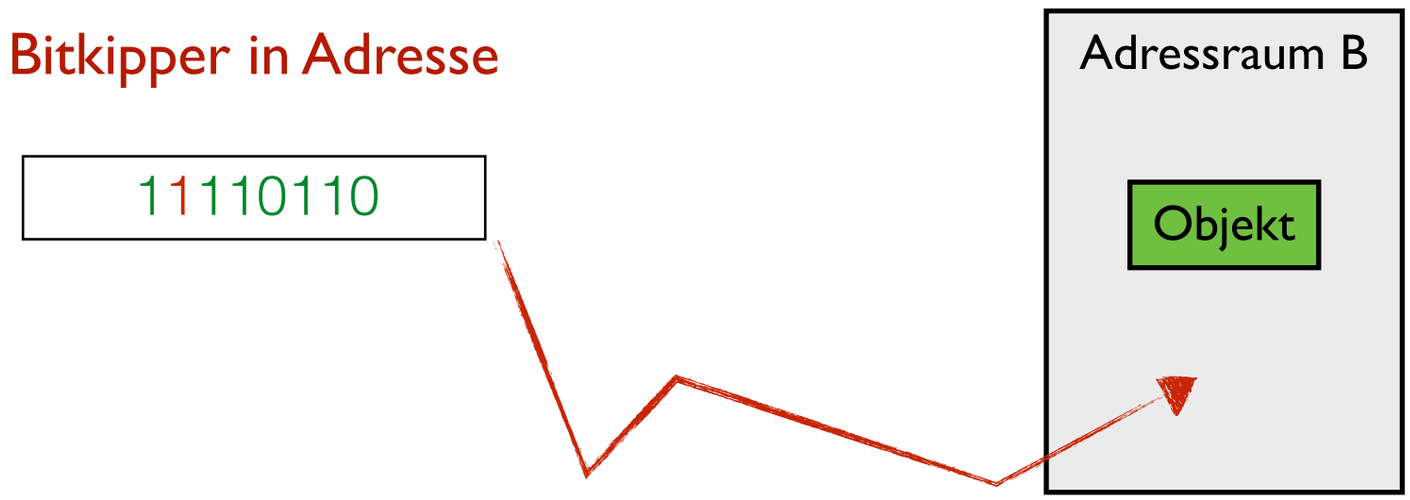
Wilde Referenzen

- Java-Referenz: Speicheradresse der Daten
- Kaputte Referenz: Wilder Zeiger
- Schlimmster Fall: Zugriff auf beliebigen Speicherbereich



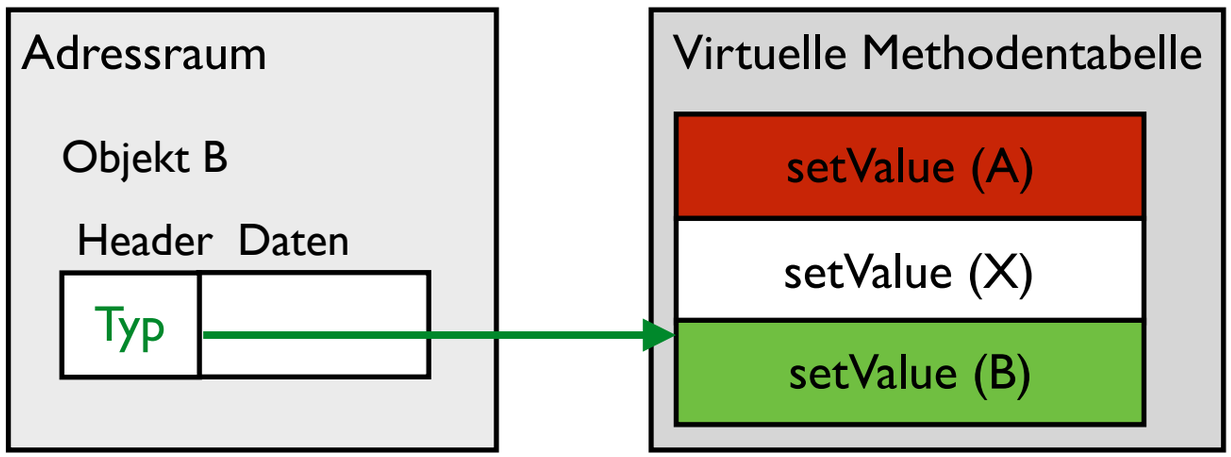
Wilde Referenzen

- Java-Referenz: Speicheradresse der Daten
- Kaputte Referenz: Wilder Zeiger
- Schlimmster Fall: Zugriff auf beliebigen Speicherbereich



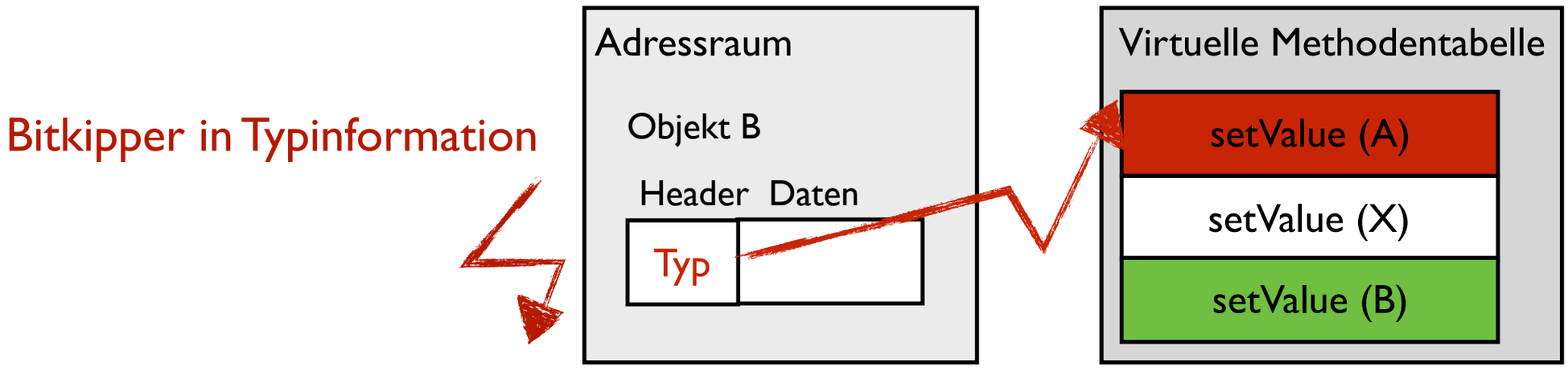
Wilde Referenzen

- Java-Referenz: Speicheradresse der Daten
- Kaputte Referenz: Wilder Zeiger
- Schlimmster Fall: Zugriff auf beliebigen Speicherbereich



Wilde Referenzen

- Java-Referenz: Speicheradresse der Daten
- Kaputte Referenz: Wilder Zeiger
- Schlimmster Fall: Zugriff auf beliebigen Speicherbereich



Wilde Referenzen

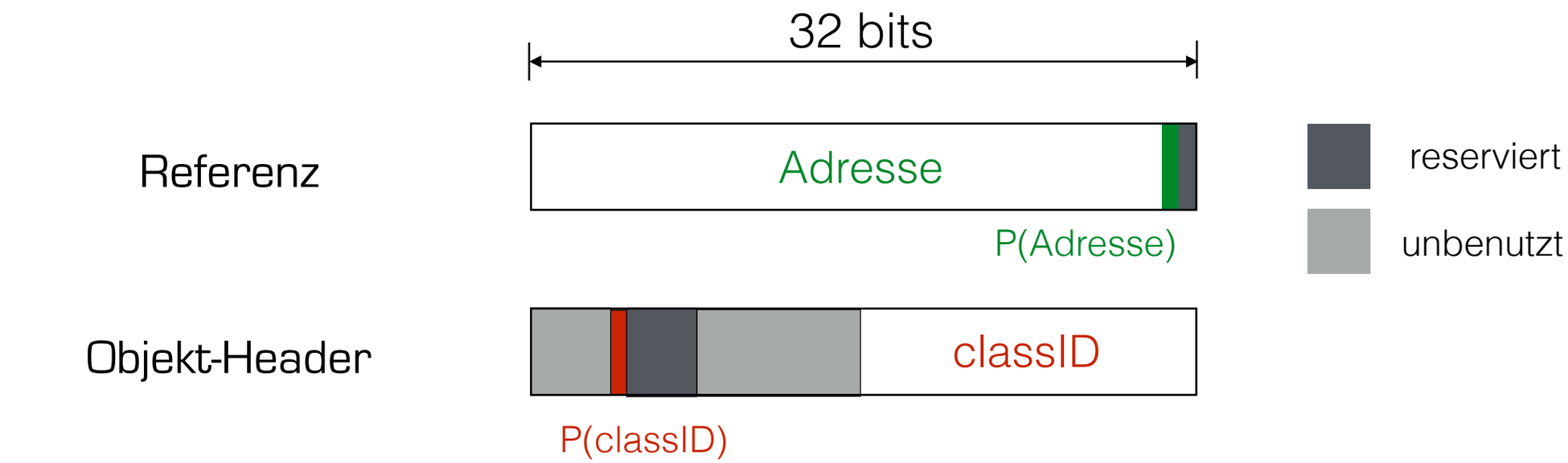
- Java-Referenz: Speicheradresse der Daten
- Kaputte Referenz: Wilder Zeiger
- Schlimmster Fall: Zugriff auf beliebigen Speicherbereich

Lösung**: Referenzintegrität** beim **Zugriff prüfen:**

z.B. durch Paritätsbits: Schutz der Adresse und der Typinformation

- Alle Referenzen sind zur Laufzeit bekannt
- Laufzeitüberprüfungen werden durch den Übersetzer eingefügt

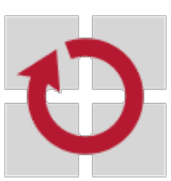
Integritätsprüfungen



- Beispielhafte Umsetzung der Referenzabsicherung
- 32-bit Plattform: TriCore TC1796 CPU, 1 MB adressierbarer Speicher
- Keine Erhöhung der Angriffsfläche

Integritätsprüfungen: Eigenschaften

+ Anpassbar durch den Entwickler
  - Zeitpunkt der Ausführung
  - Umfang der Redundanzinformation
+ Effizient und vorhersagbar bezüglich Speicherverbrauch
  - Kein Mehrbedarf an Speicher, keine Erhöhung dessen Angriffsfläche
  - Alignment und Adresseigenschaften des Mikrocontrollers
+ Vorhersagbar bezüglich Rechenzeit: Konstanter Mehraufwand
- Verursacht jedoch einen Anstieg des Rechenzeitbedarfs
  - Erhöhung der Angriffsfläche bezüglich Zeit

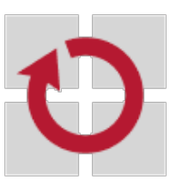


Randbedingungen

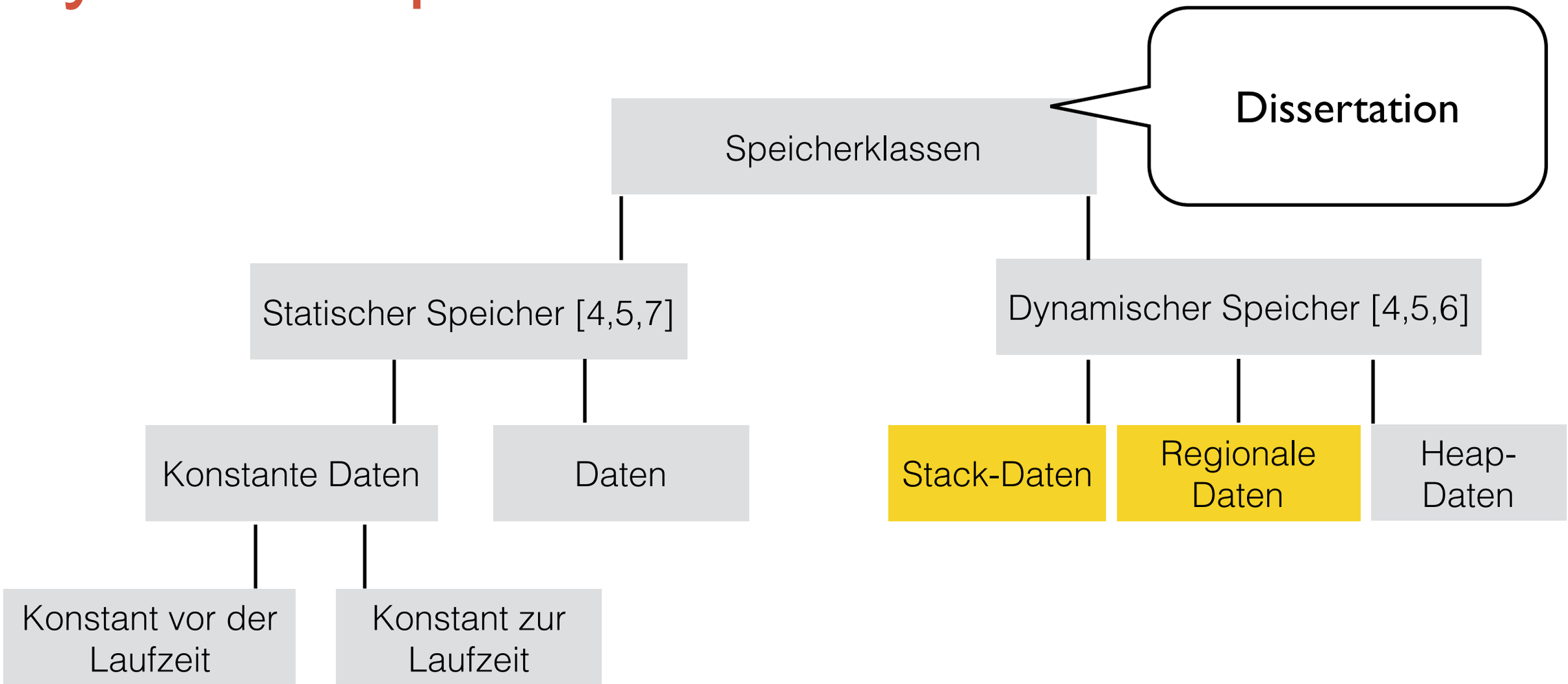
- **Verlässlichkeit** in Anwesenheit transienter Fehler?
  - Einfügen von Integritätsprüfungen
  - Vermeidung von Fehlern
- **Ressourceneffizienz** und **Vorhersagbarkeit**?
  - Anwendungsspezifisches Speichernutzungsverhalten
  - Automatisierte Bestimmung und Verwaltung von Speicherklassen

Effizienz, Vorhersagbarkeit, Verlässlichkeit

- Effizienz, Vorhersagbarkeit und Verlässlichkeit sind verwoben
- Ausführungszeit und Speicherverbrauch
- Adressierung der Eigenschaften
  - Anwendungsspezifische Speichernutzungsverhaltens
  - Automatisierung durch Escape-Analyse und Garbage Collector



Hybrides Speichermodell



JTRES ‘10 [4] Stilkerich et al: KESO: An Open-Source Multi-JVM for Deeply Embedded Systems
CPE ’12 [5] M. Stilkerich, **I. Stilkerich**, C. Wawersich, W. Schröder-Preikschat: Tailor-Made JVMs for Statically Configured Embedded Systems
JTRES ’14 [6] Stilkerich et al: Fragmentation-Tolerant Real-Time Memory Management Revisited
JTRES ’14 [7] C. Erhardt, S. Kuhnle, **I. Stilkerich**, W. Schröder-Preikschat: The final Frontier: Coping with Immutable Data in a JVM for Embedded Real-Time Systems

Escape-Analyse

- Automatisches Herleiten von stack-allokierbaren Objekten
  - Verlässliches Erkennen von Referenzen
  - Nutzung von Informationen aus Alias-Analyse
  - Erreichbarkeit von Referenzen

Escape-Analyse

- Vorteile der Stack-Allokation
  - Verwaltung von Objekten ist effizient und vorhersagbar
  - Verursacht keine Fragmentierung des Speichers
  - Keine Synchronisation von Stack-Objekten nötig
- Co-Design: Escape-Analyse von Choi et al. (TOPLAS ‘03)

Anwendungen der Escape-Analyse

Weitere Anwendungsgebiete [8,9]

1. Implementierung thread-lokalen Speichers
2. Vermeidung der Auswirkung transienter Fehler durch Kurzlebigkeit
3. Effizienzsteigerung von Fehlertoleranzmaßnahmen
4. Effiziente Kommunikation von software-isolierten Komponenten
5. Scope-Erweiterung von Objektvariablen
6. Baustein einer Analyse zur Herleitung konstanter Daten
7. Verbesserung der Vorhersagbarkeitsanalyse eines Garbage Collectors

LCTES '15 [8] Stilkerich et al: A Practical Getaway: Applications of Escape Analysis in Embedded Real-Time Systems
TECS '17 [9] Stilkerich et al: The Perfect Getaway: Escape Analysis in Embedded Real-Time Systems

Anwendungen der Escape-Analyse

Weitere Anwendungsgebiete [8,9]

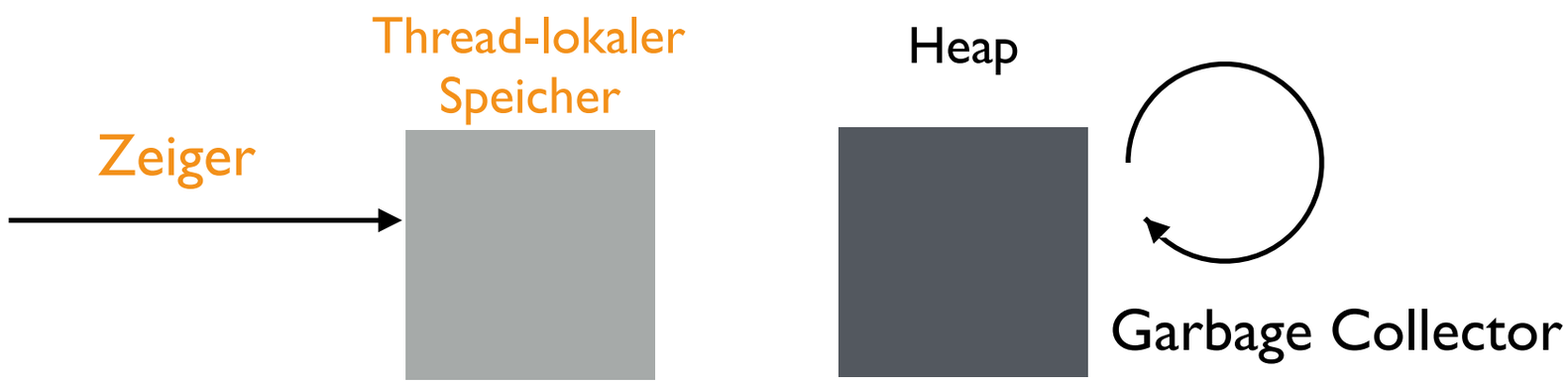
1. Implementierung thread-lokalen Speichers
2. Vermeidung der Auswirkung transienter Fehler durch Kurzlebigkeit
3. Effizienzsteigerung von Fehlertoleranzmaßnahmen
4. Effiziente Kommunikation von software-isolierten Komponenten
5. Scope-Erweiterung von Objektvariablen
6. Baustein einer Analyse zur Herleitung konstanter
7. Verbesserung der Vorhersagbarkeitsanalyse eines Garbage Collectors

Dissertation

LCTES ’15 [8] Stilkerich et al: A Practical Getaway: Applications of Escape Analysis in Embedded Real-Time Systems
TECS ’17 [9] Stilkerich et al: The Perfect Getaway: Escape Analysis in Embedded Real-Time Systems

Thread-lokaler Speicher

- Spezieller Speicherbereich für eine Klasse von Objekten
  - Von einem bestimmten Thread erreichbar
- Koexistenz mit Heap (Garbage Collector) möglich

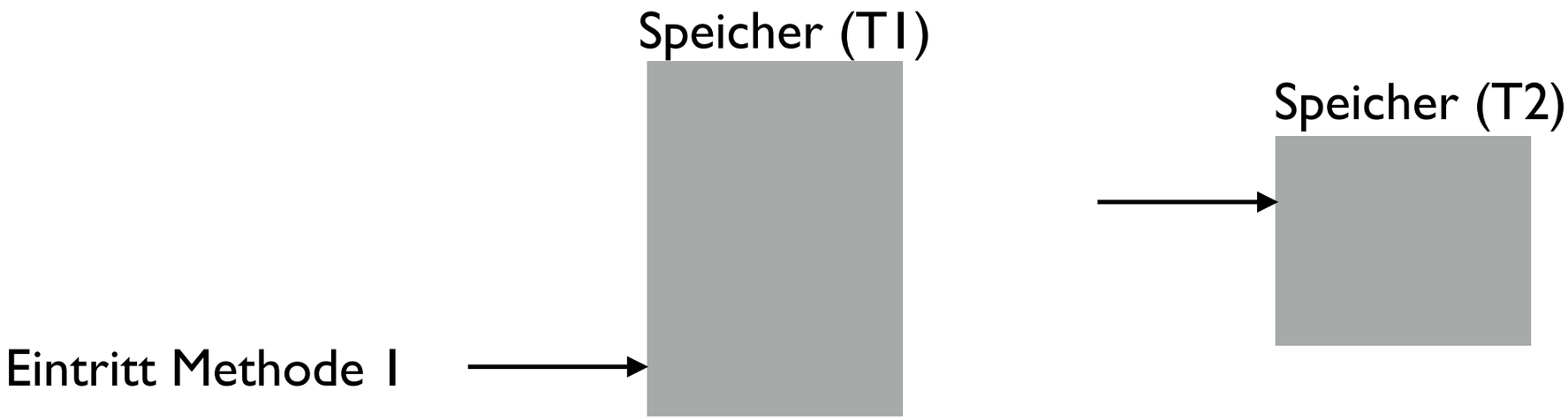


Thread-lokaler Speicher: Beispiel

- Übersetzergestützte Verwaltung des Speicherbereichs
- Jeder Methode ist eine logische Region zugeordnet
  - Regionen sind ähnlich eines Stacks organisiert
- Jeder Thread besitzt einen eigenen Speicherbereich

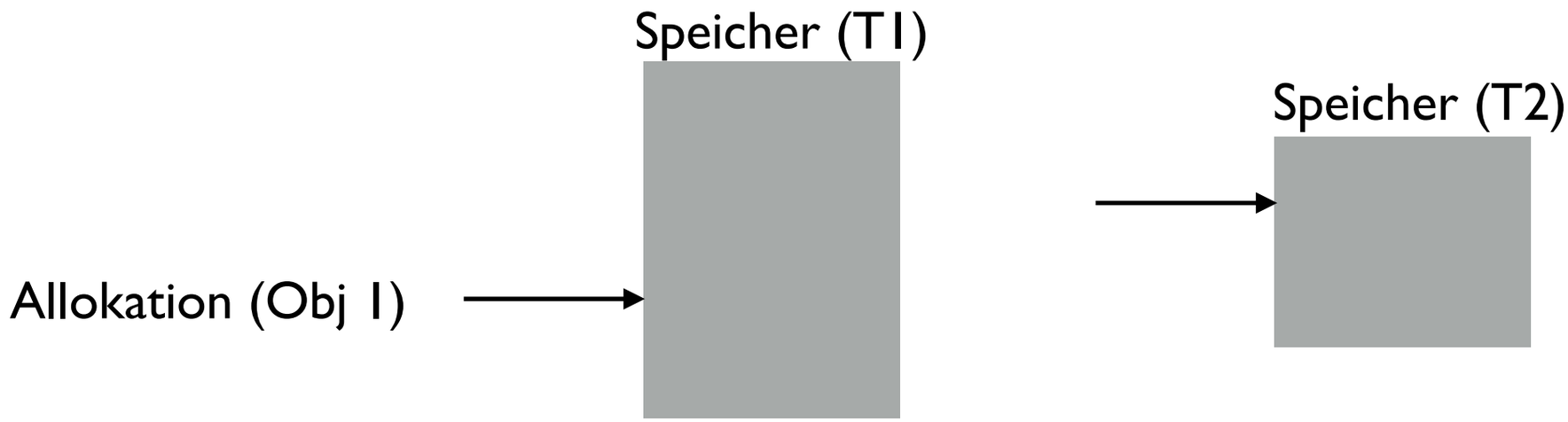
Thread-lokaler Speicher: Beispiel

- Jeder thread-lokale Speicher besitzt
  - Füllstandszeiger
  - Maximales Befüllungsniveau



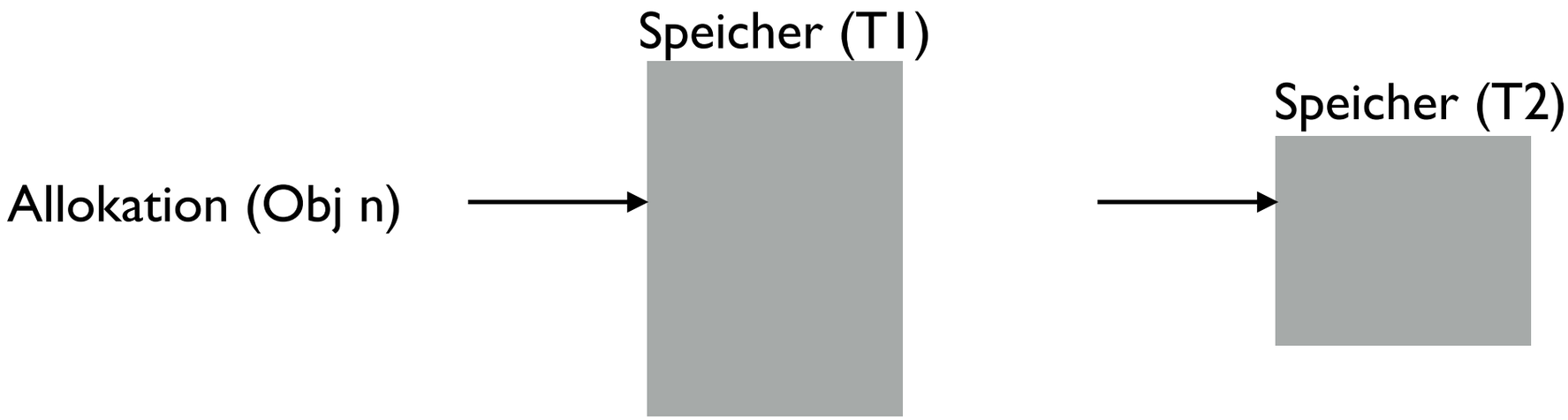
Thread-lokaler Speicher: Beispiel

- Jeder thread-lokale Speicher besitzt
  - Füllstandszeiger
  - Maximales Befüllungsniveau



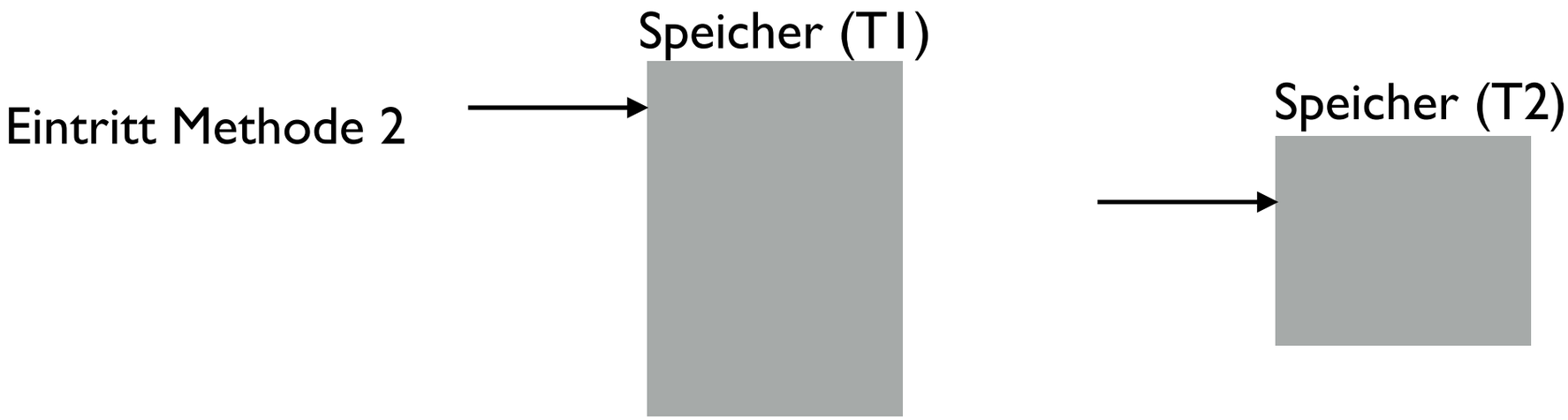
Thread-lokaler Speicher: Beispiel

- Jeder thread-lokale Speicher besitzt
  - Füllstandszeiger
  - Maximales Befüllungsniveau



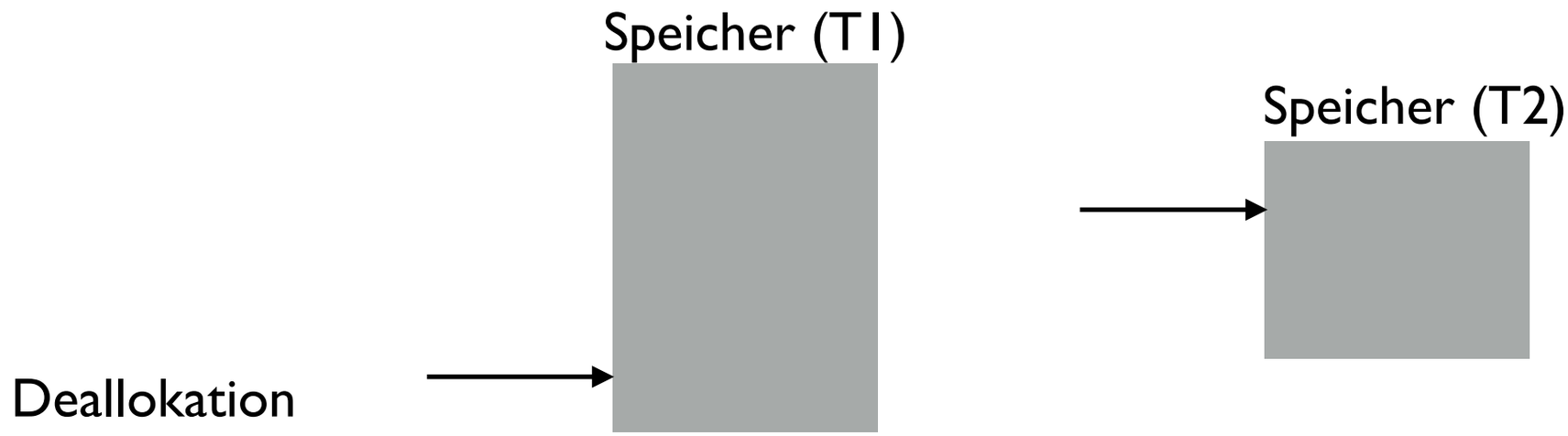
Thread-lokaler Speicher: Beispiel

- Jeder thread-lokale Speicher besitzt
  - Füllstandszeiger
  - Maximales Befüllungsniveau



Thread-lokaler Speicher: Beispiel

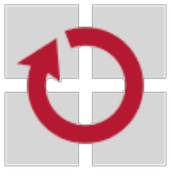
- Jeder thread-lokale Speicher besitzt
  - Füllstandszeiger
  - Maximales Befüllungsniveau



Randbedingungen

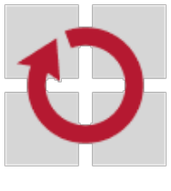
- **Verlässlichkeit** in Anwesenheit transienter Fehler?
  - Einfügen von Integritätsprüfungen
  - Vermeidung von Fehlern

- **Ressourceneffizienz** und **Vorhersagbarkeit**?
  - Anwendungsspezifisches Speichernutzungsverhalten
  - Automatisierte Bestimmung und Verwaltung von Speicherklassen

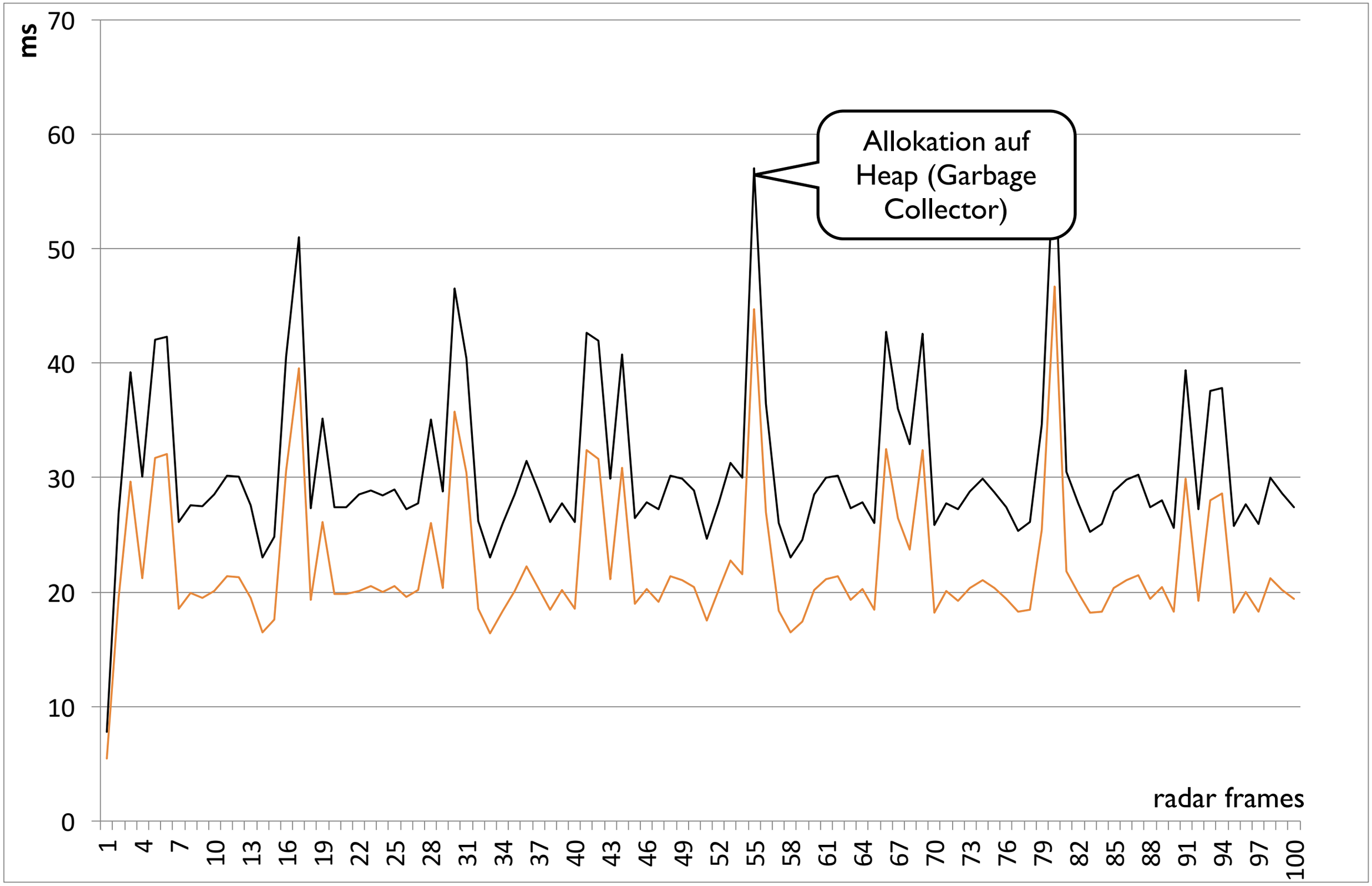


Evaluation

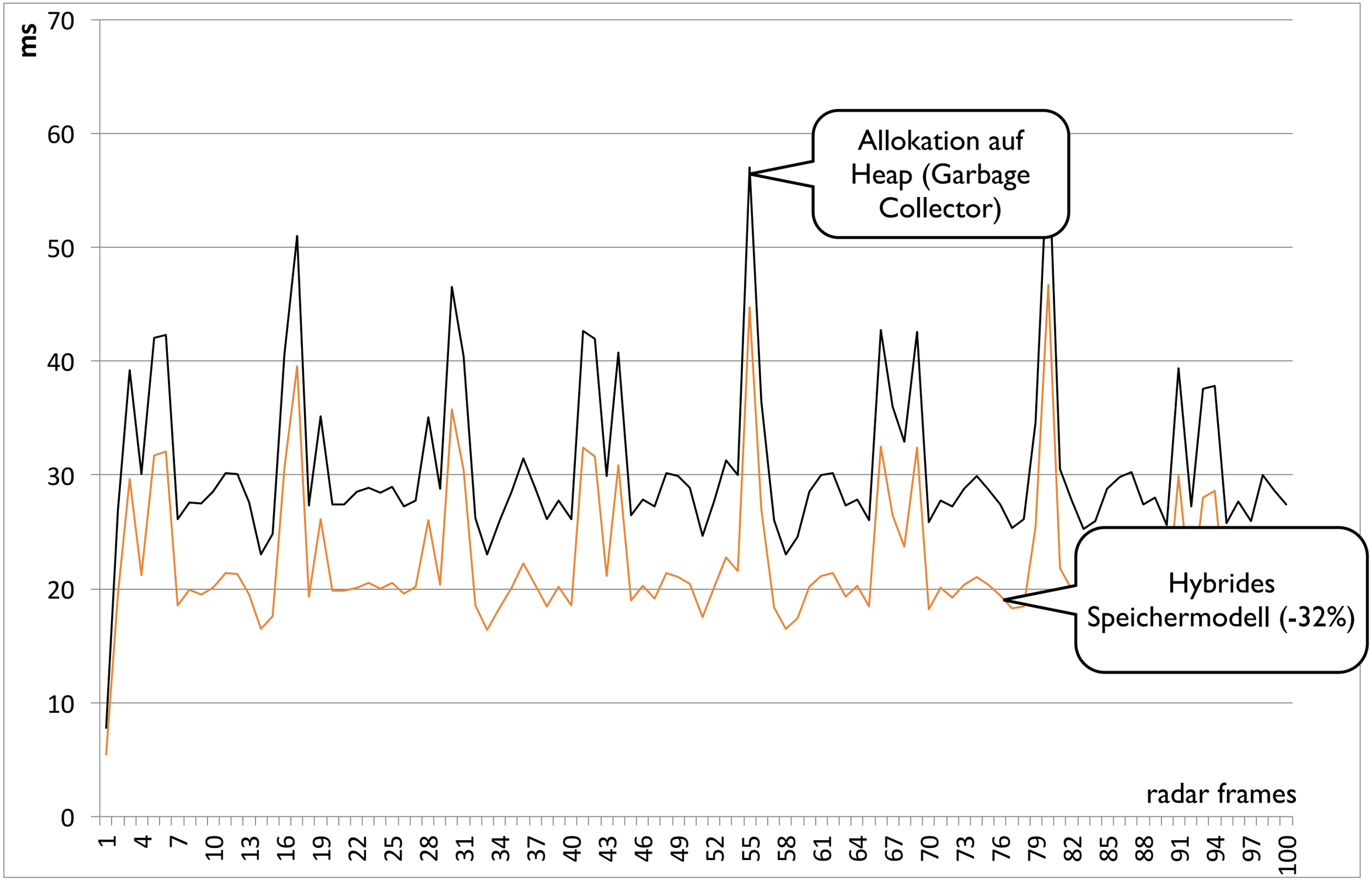
- Benchmark für Echtzeitsysteme *Collision Detector (CD_x)*
- Vorgestellte CMM-Varianten (Java)
  - Nur Garbage Collector
  - Hybrides Speichermodell
  - Integritätsüberprüfungen
- Untersuchung der Implementierung: Realitätsnahe Bedingungen
  - AUTOSAR OS
  - TriCore TC1796 (32-bit Prozessor)
    - 1 MiB externer SRAM, 2 MiB interner Flashspeicher
    - CPU (Taktung: 150MHz, 75 MHz Bus)



Laufzeitmessungen



Laufzeitmessungen

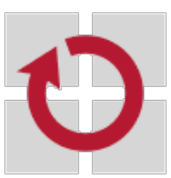


Laufzeitmessungen

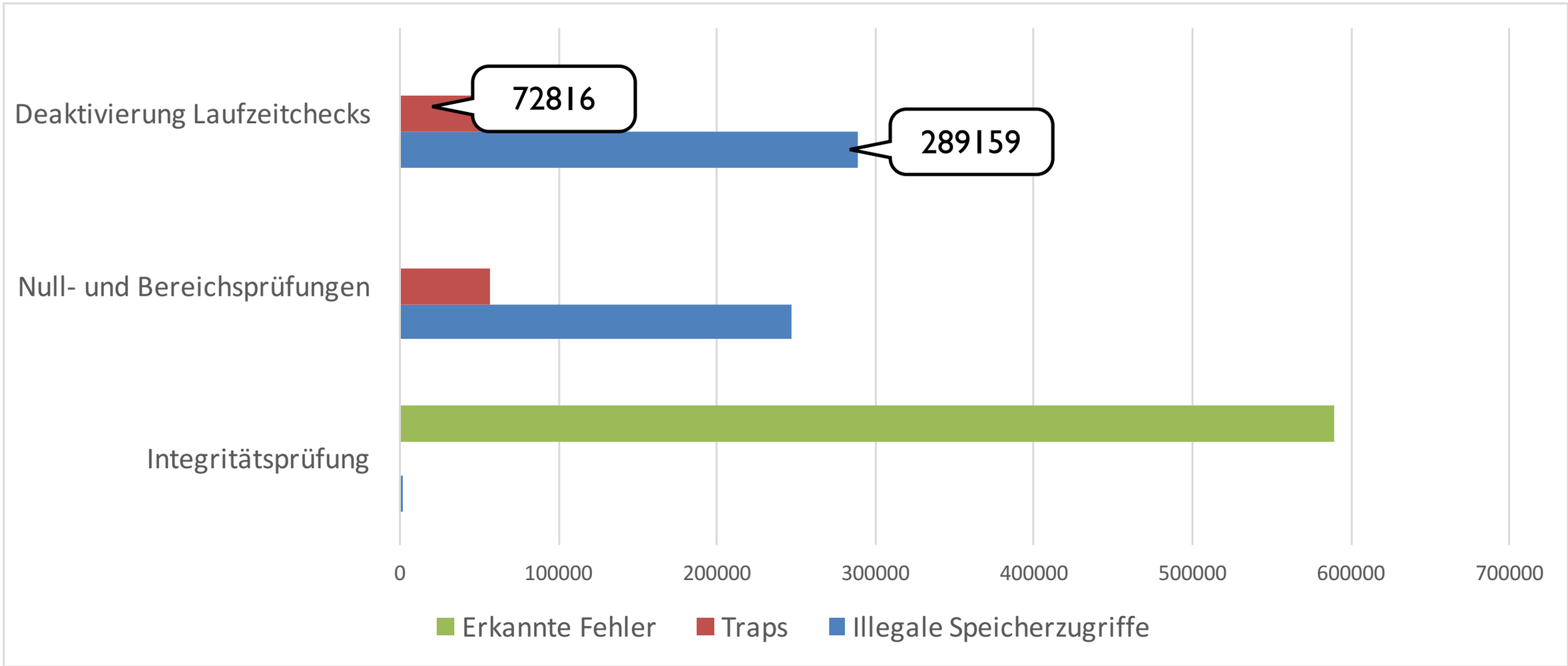
- Hybrides Speichermodell mit Echtzeit Garbage Collector weist einen mittleren Mehraufwand von 12% im Vergleich zur C-Variante auf
- Ohne Garbage Collector: Im Mittel 6% schneller als C-Variante

Effektivität der Integritätsprüfung

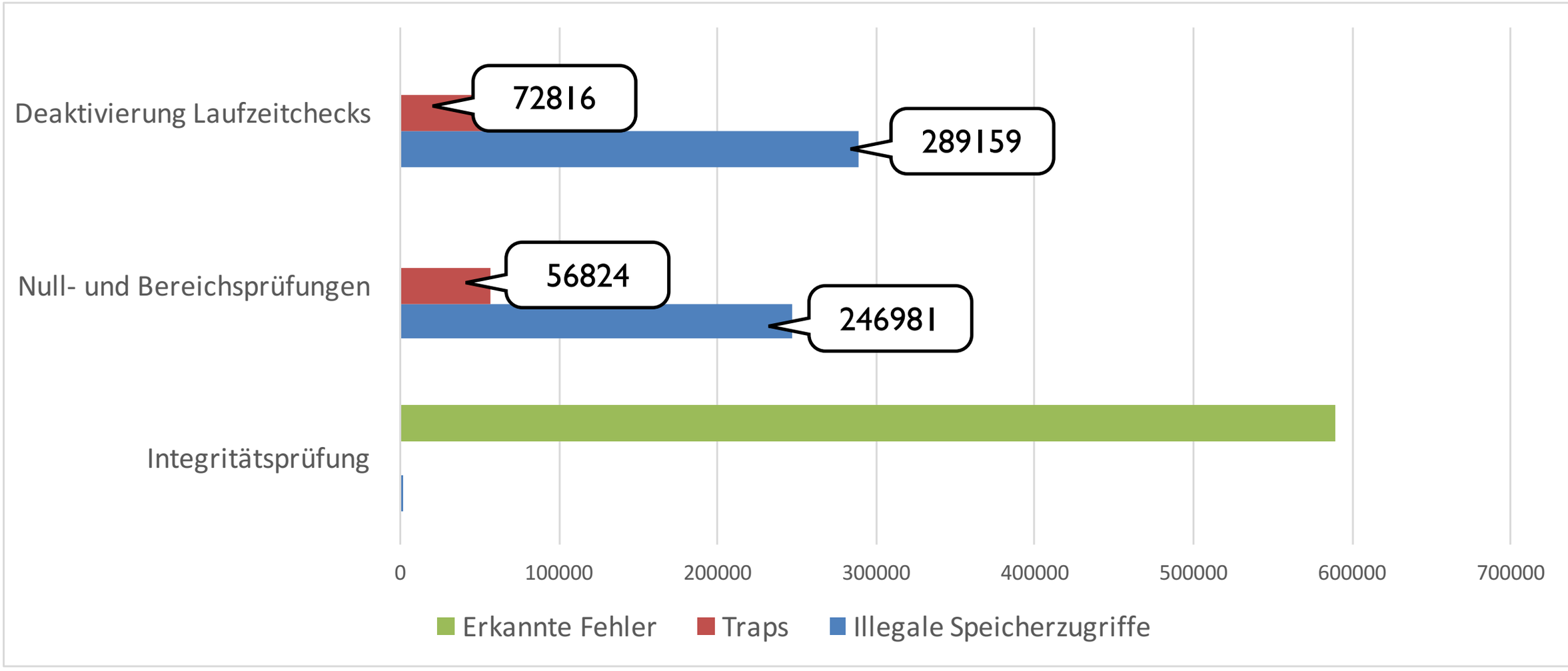
- Analyse mit dem FAIL*-Werkzeug
- Injizierung von 1-Bit-Fehlern in jede Position eines Speicherworts
- Gruppierte Anwendungsdaten: KESOs Erreichbarkeitsanalyse



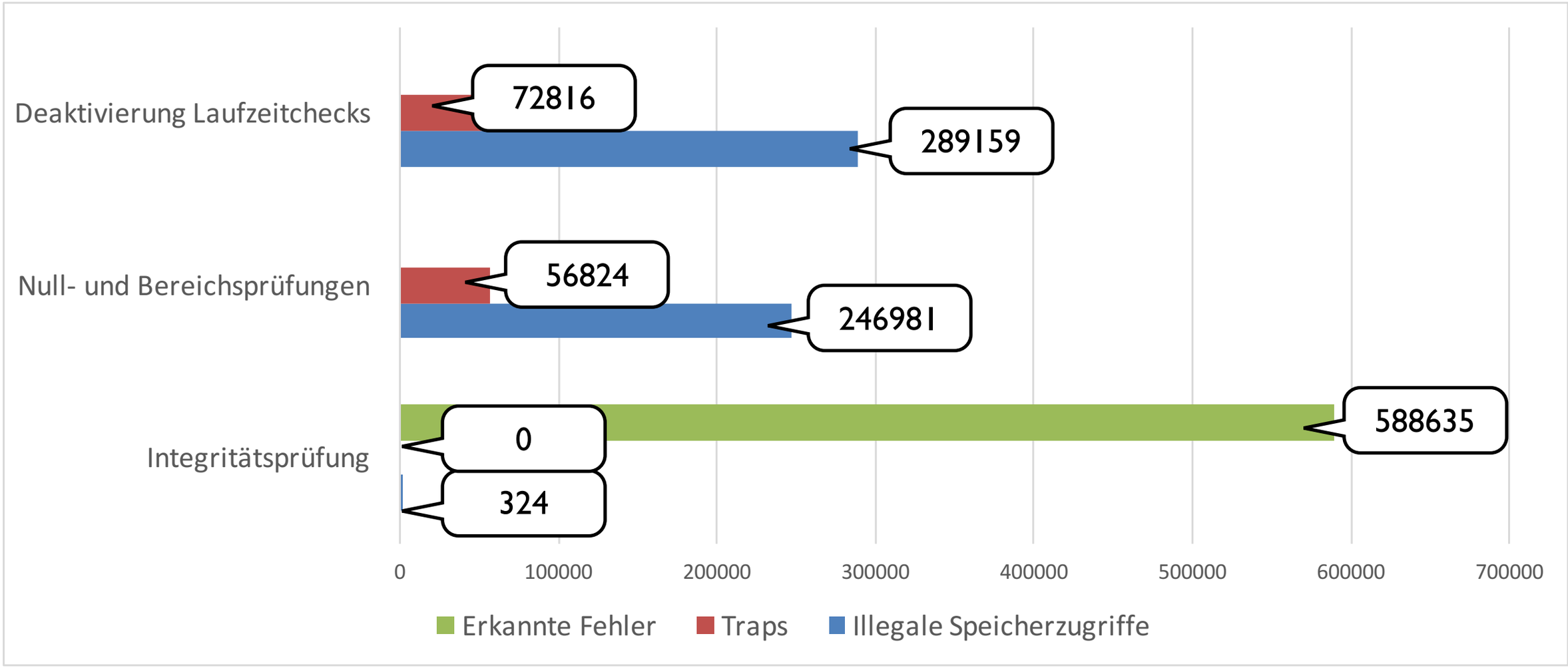
Effektivität der Integritätsprüfung



Effektivität der Integritätsprüfung



Effektivität der Integritätsprüfung



Effektivität der Integritätsprüfung

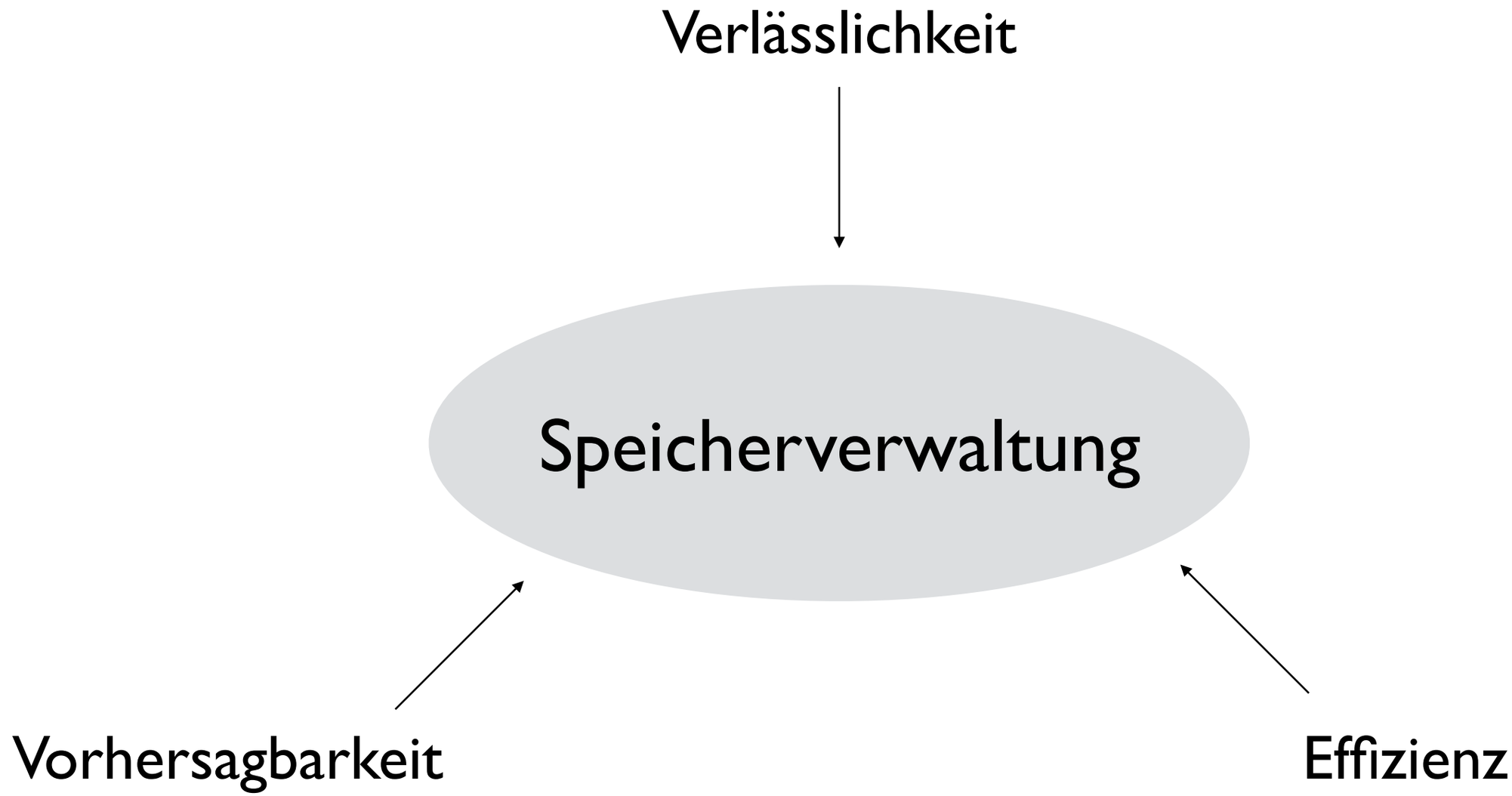
Effektivität der Integritätsprüfung

- Isolationseigenschaft ausreichend sichergestellt
- Verlässliche, automatische Speicherverwaltung möglich

Integritätsprüfungen

- Einfluss auf die mittlere Laufzeit
  - Mehraufwand in Anwendung (nur Referenzen): 30,7%
- Absicherung des Garbage Collectors
  - Referenzprüfung: 44,2%
  - Ausgewählte Verwaltungsinformation: 38,6%
- Rechenzeitbedarf des Garbage Collector vergleichsweise klein
  - Integritätsprüfungen verursachen vorhersagbaren Mehraufwand
- Kein Mehrbedarf an Speicher zur Laufzeit

Co-Design: Randbedingungen



Co-Design: Randbedingungen

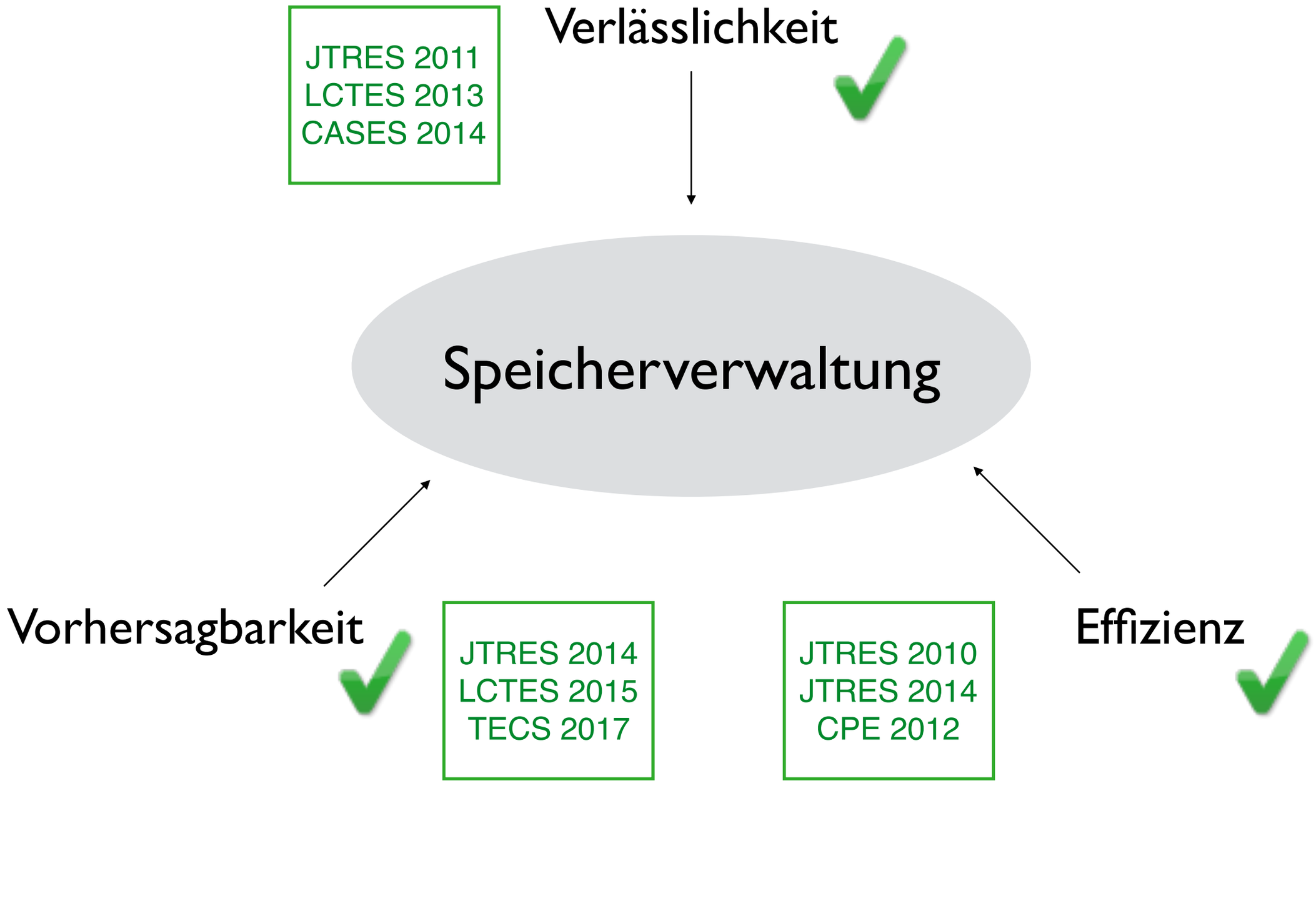
JTRES 2011
LCTES 2013
CASES 2014

Verlässlichkeit

Vorhersagbarkeit

Effizienz

Co-Design: Randbedingungen



Forschungsfrage

Wie kann eine **maßgeschneiderte Speicherverwaltung** für **Anwendungsprogramm** und die verwendete **Hardware** bereitgestellt werden?

Co-Design der Speicherverwaltung

- Verlässlichkeit, Effizienz, Vorhersagbarkeit
- Berücksichtigung anwendungsspezifischen Speichernutzungverhaltens
- Kooperation mit dem Entwicklern
- Automatisiert durch Unterstützung des Laufzeitsystems