# Einleitung

* Warum ist Concurrency relevant?
* Warum spielen Concurrency Bugs eine Rolle?
* Warum ist die Erkennung von Deadlocks so wichtig?
* Was wird in der Arbeit gemacht?
  + Erläuterung von Concurrency Bugs und Deadlocks im Besonderen
  + Einführung in die Lock Dependency Methode
  + Sync-Preserving Deadlocks und warum diese die häufigsten Deadlocks in der Praxis sind; warum diese effizient erkannt werden können
  + Beschreibung SPDAlgorithmus in Online und Offline

Sei es in Betriebssystemen, Web-Servern oder Echtzeitsystemen – in fast jeder modernen Applikation ist die Nebenläufigkeit (engl. „Concurrency“) von Tasks von großer Bedeutung. Beispielsweise könnte ohne Nebenläufigkeit ein Betriebssystem nicht mehr als ein Programm gleichzeitig laufen lassen. Ein IoT-Gerät müsste für die Netzwerkkommunikation die Aufnahme von Sensordaten stoppen. Allein diese Beispiele zeigen die Relevanz von Nebenläufigkeit.

Nichtsdestotrotz gibt es neben den Vorteilen der Nebenläufigkeit, wie z.B. dem Performancegewinn durch das Verteilen von Tasks auf mehreren Prozessorkernen, auch Probleme, die mit ihr einhergehen – sog. „Concurrency Bugs.“ Grundlage dieser Concurrency Bugs sind die nebenläufigen Zugriffe verschiedener Tasks auf dieselben Ressourcen. Diese Zugriffe können dazu führen, dass ein Programm, welches die Tasks nebenläufig ausführt, abstürzt oder sogar in einer Endlosschleife verweilt, ohne tatsächlich Arbeit zu verrichten.

Es gibt mehrere Typen von Concurrency Bugs. Der wohl bekannteste von allen ist der sog. „Deadlock“. Dieser kommt zustande, wenn zwei Threads bereits eine Ressource reserviert haben, die der jeweils andere zum gleichen Zeitpunkt ebenso reservieren will. Wenn ein Deadlock vorkommt stürzt das Programm in Folge ab. Die Forschung beschäftigt sich schon sehr lange damit, Deadlocks zuverlässig und effizient zu erkennen bzw. vorherzusagen. Dabei wurden statische und dynamische Lösungsansätze entwickelt. Statische Lösungsansätze versuchen einen Deadlock anhand des Quellcodes zu erkennen, wohingegen dynamische Ansätze die Ausführung des Programms analysieren. Probleme dieser Lösungsansätze waren jedoch häufig, dass sie entweder zu viele false positives (Deadlocks, die aber keine sind) angezeigt oder eine zu schlechte Laufzeit haben.

Die vorliegende Arbeit beschäftigt sich jedoch mit einem kürzlich erbrachten und bemerkenswerten Fortschritt in der Vorhersage von Deadlocks. Nachdem in den nachfolgenden Kapiteln zuerst auf Concurrency Bugs und Deadlocks im Besonderen eingegangen wird, werden Methoden vorgestellt, die es ermöglichen in linearer Laufzeit sowie mit sehr hoher Präzision Deadlocks vorherzusagen.