Entwicklung und Evaluation eines

Sitzungsservers

für das

SAROS-Projekt

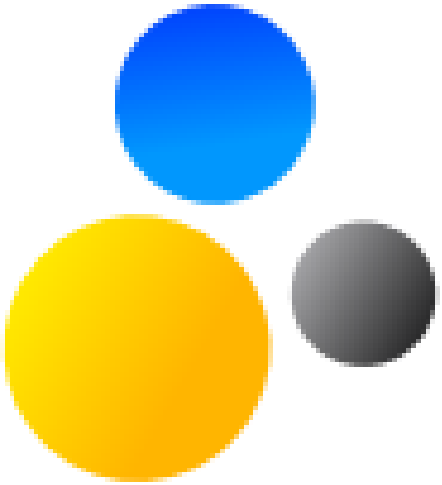
Masterarbeit

Denis Washington

Betreuer: **Franz Zieris**

Gutachter: **Prof. Lutz Prechelt, Prof. Adrian Paschke**

SAROS – Distributed Party Programming



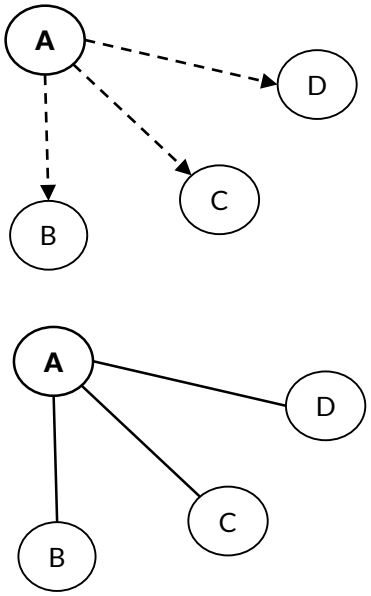
- Verteilte Echtzeitkollaboration zwischen Entwicklern
  - Paarprogrammierung, Code Reviews, …
- Plugin für Eclipse (Saros/E) und IntelliJ (Saros/I)
  - Saros/I noch nicht veröffentlicht

Dieser Vortrag

- Motivation und Ziele
- Umsetzung & Ergebnisse
- Zusammenfassung

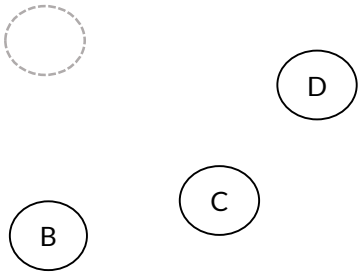
Motivation und Ziele

Saros hat ein **host-basiertes Sitzungsmodell**



Initiator einer Sitzung ist dessen **Host**

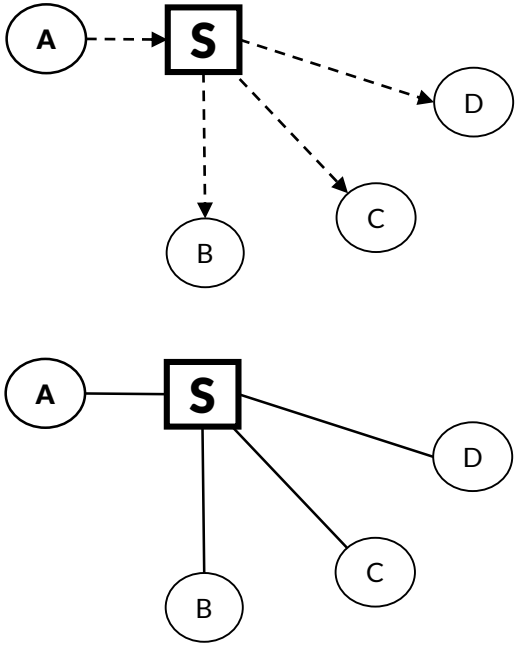
- Sonderrechte (Projekt teilen, Teilnehmer einladen)
- Koordinator (Knotenpunkt für Kommunikation)
- “Quelle der Wahrheit” (definiert Sitzungszustand)



Sitzung kann ohne Host nicht bestehen

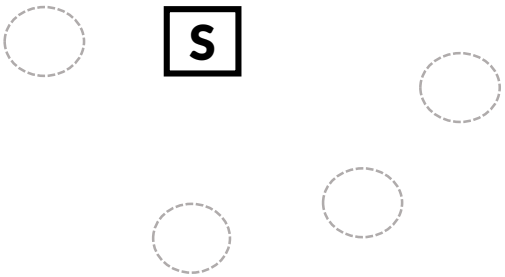
- Verlässt er die Sitzung, ist diese beendet
- Verhindert (teil-)asynchrone Kollaboration
  - B kann nicht zwischenzeitlich Sitzung verlassen und später aktuellen Stand des Codes abholen, wenn A bis dahin gegangen ist

Idee: **Server-basierte Sitzungen**



Sitzungsserver übernimmt Host-Rolle

- Übernimmt Sitzungsaufbau für Initiator
- Koordinator, “Quelle der Wahrheit”

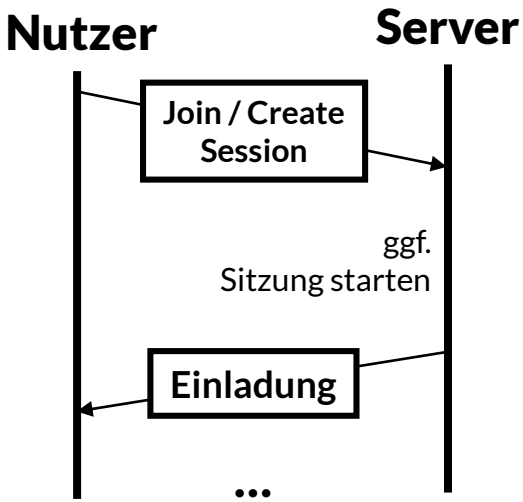


Kann ohne Teilnehmer weiterlaufen

- Ermöglicht langlebige Sitzungen
- Teilnehmer müssen nicht gleichzeitig anwesend sein

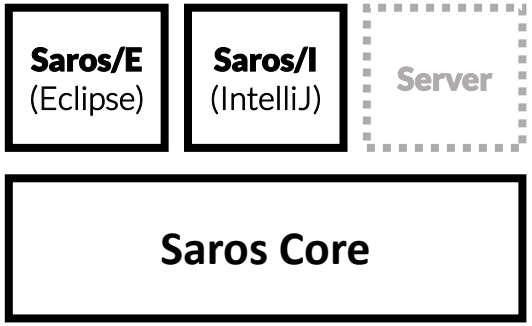
Vorarbeit: **Nils Bussas (2014)**

- Bachelorarbeit: **„Entwicklung eines Server-Prototypen für Saros“**
- Protokoll zur Sitzungsinitiierung auf Server
  - **Server Session Request:** Nutzer kann um Sitzungserstellung (“*Create Session*”) oder -beitritt (“*Join Session*”) bitten; Server verschickt Sitzungseinladung an Nutzer
- Prototyp-Umsetzung in Saros/E (Eclipse)
- Einschränkungen:
  - **Keine Möglichkeit, Projekte zu teilen**
  - Server-Prototyp hängt von Eclipse ab
  - Nur eine Sitzung pro Server



Vorarbeit: **Saros-Kern**

- Wiederverwendbaren, IDE-unabhängige Teile von Saros
  - Netzwerkkommunikation, Sitzungslogik, Dateisynchronisation, …
- Weniger Duplikation zwischen Saros/E und /I
  - siehe Bachelorarbeit von Arndt Lasarzik (2015)
- Als Grundlage für Sitzungsserver nutzbar
  - Allerdings fehlten noch Teile der Grundfunktionalität (u.a. Projektverteilung, Sitzungsverwaltung)



Ziele der Arbeit

- Saros-Protokoll für Server-Sitzungen vervollständigen
  - Projektteilen als Nicht-Host
- Eigenständigen Sitzungsserver entwickeln
  - Migration fehlender Grundfunktionalität in den Kern
  - Server-spezifische Implementation von Kern-Schnittstellen
- Server und Saros-Kern auf Ressourcenlecks untersuchen
  - **Laufzeitanalyse** mit automatisierten Stresstests
  - **Statische Analyse** für Erkennung einfacher Ressourcenlecks

Nicht-Ziele

- Vollständiger Funktionsumfang
  - z.B. mehr als eine Sitzung
- Produktionsreife
  - z.B. Aufhebung der Eine-Sitzung-pro-Server-Beschränkung
  - Aber: Dokumentation bekannter zu behebender Probleme

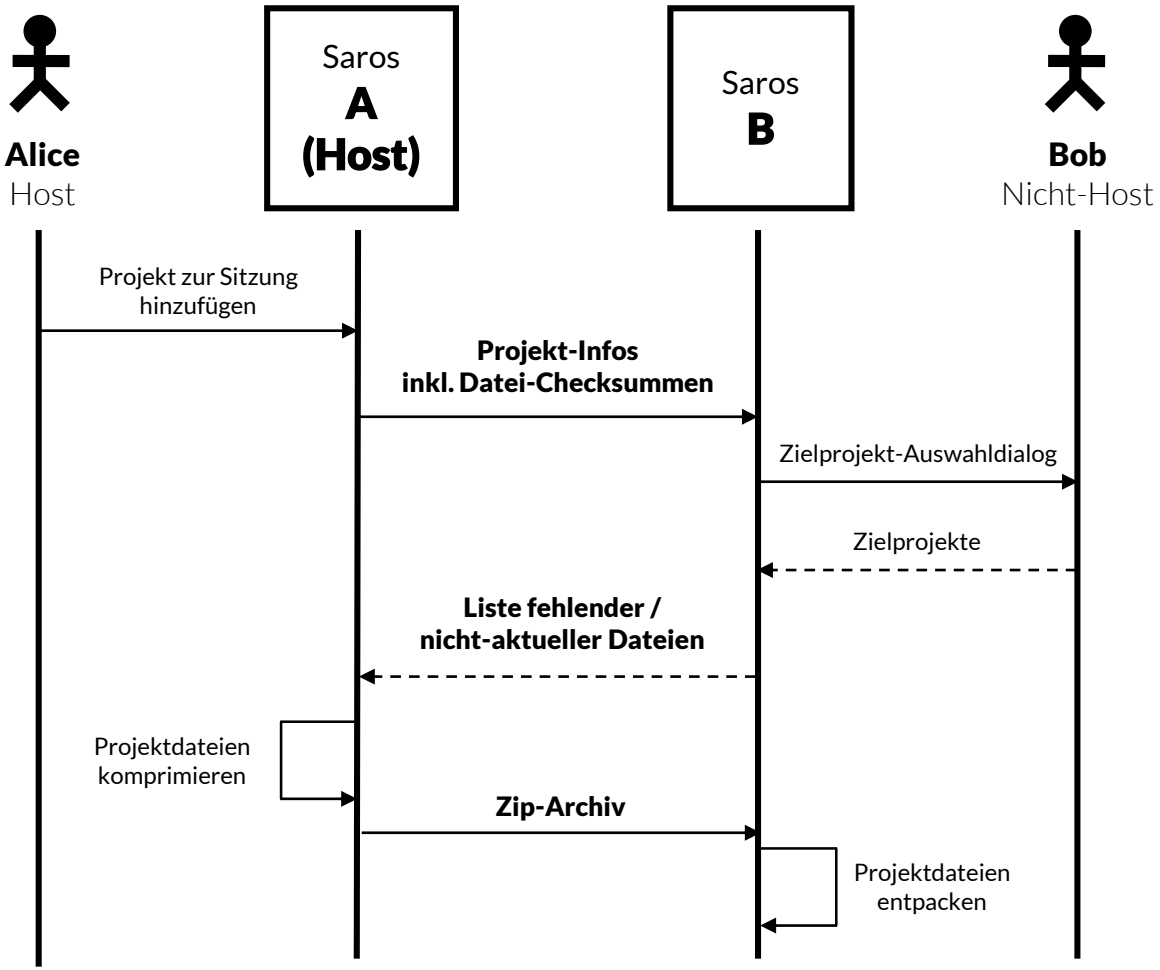
Umsetzung & Ergebnisse

Non-Host Project Sharing

in Zusammenarbeit mit Ute Neise

- **Problem:** Projekte dürfen nur vom Host geteilt werden
  - **In server-basierten Sitzungen schlecht:** Host nur eine Maschine
  - Beschränkung verhindert Koordinations- und Konsistenzprobleme (Starroske, Schlott)
- **Lösungsansatz:** Nicht-Host kann Host um Teilen eines Projekts bitten
  - Nicht-Host schickt Projekt an Host
  - Host teilt es mit anderen Teilnehmer

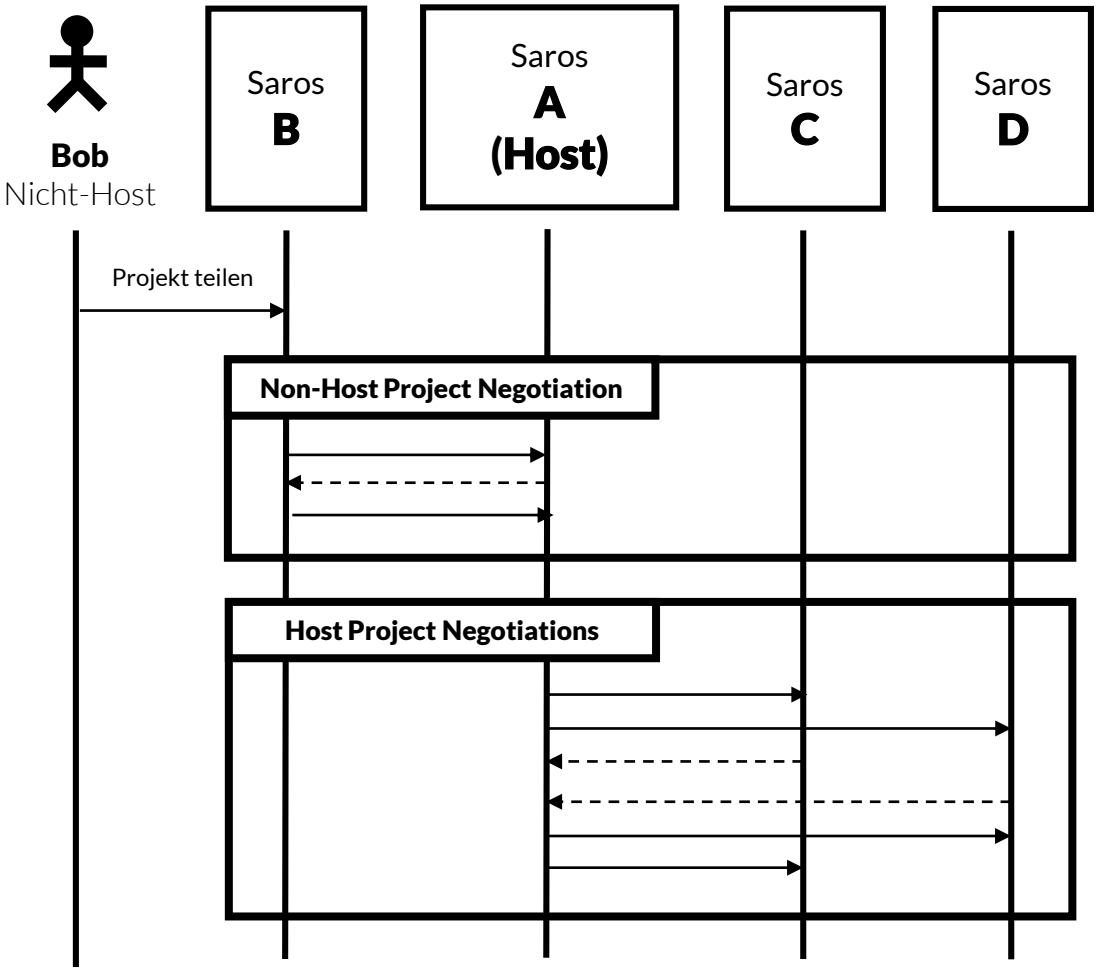
Der Projektaustausch: **Project Negotiation**



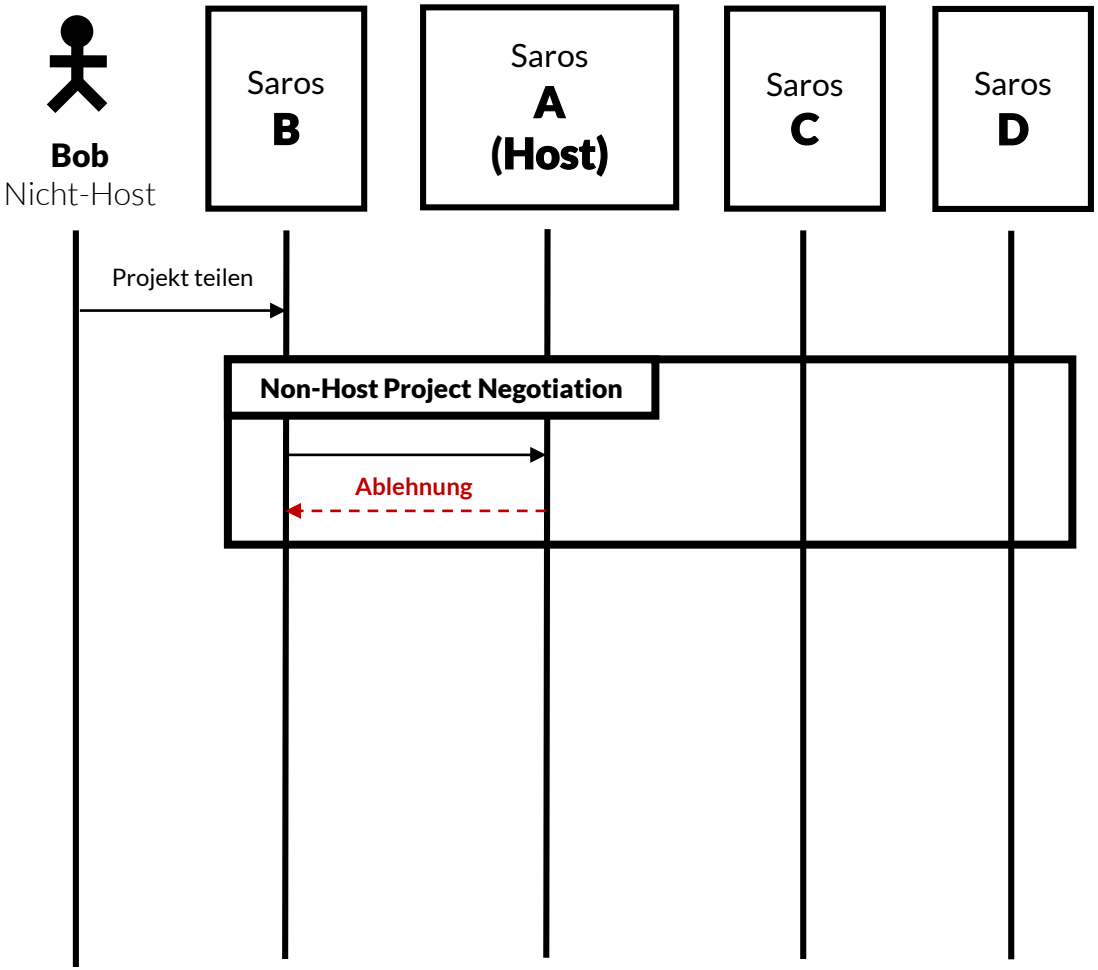
Non-Host Project Sharing: **Anforderungen**

- **Wiederverwendung der Project Negotiation** so weit wie möglich
  - Weniger neue Codepfade, Potenzial für Defekte
- **Übertragbarkeit** auf klassische **host-basierte Sitzungen**
  - Größerer Nutzen
- **Keine** neuen **Konsistenz- und Nebenläufigkeitsprobleme**
  - Ursprünglicher Grund, warum Projektteilen durch Nicht-Hosts verboten

Non-Host Project Sharing: **Finale Version**



Non-Host Project Sharing: **Finale Version**



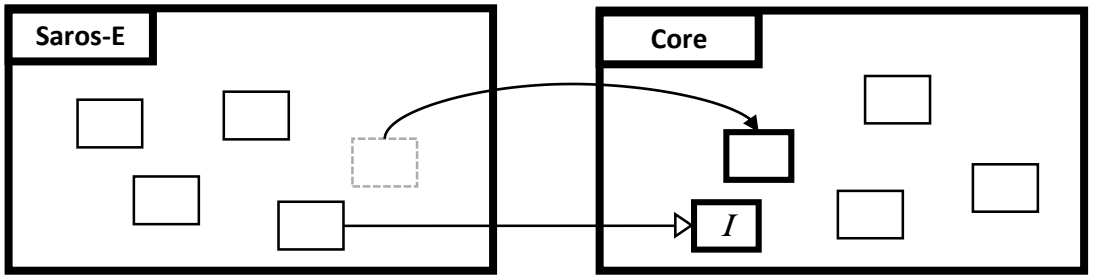
Non-Host Project Sharing: **Stärken & Schwächen**

- **+ 1:1-Wiederverwendung der Project Negotiation**
  - Keine neuen Nachrichtentypen
  - Wenig zusätzlicher Code nötig
  - Gleiche Konsistenzgarantien
- **+ Nicht spezifisch für server-basierte Sitzungen**
- **- Langsamer**
  - Nicht-Host-zu-Host- und Host-zu-Andere-Übertragung nicht parallel
  - Lösungsansatz: “Projekt-Streaming”

Non-Host Project Sharing: **Implementation & Status**

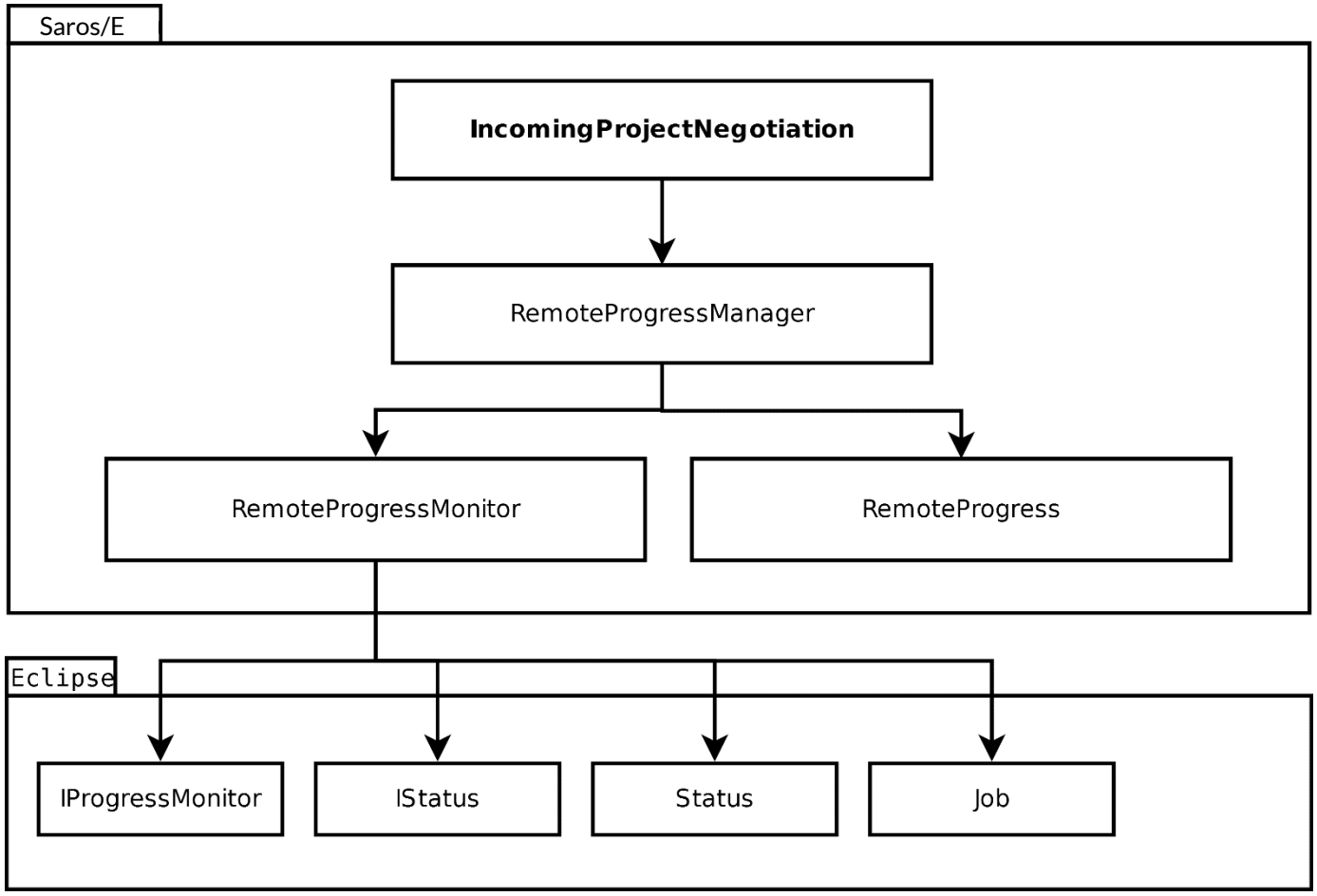
- Schwierigkeiten
  - Project Negotiation im Detail nicht dokumentiert → Reverse Engineering
  - Project-Negotiation-Code schwer zu verstehen (insb. Seiteneffekte)
- Implementation als **Change #2264** im Code-Review-System (Gerrit)
  - **+295, -65** Codezeilen
  - Nach manuellem Testplan getestet (auch mit Bussas-Server-Prototyp)
  - Noch im Review

Erweiterung des SAROS-Kerns



- **Essenzielle Nicht-Kern-Klassen identifizieren**
  - essenziell = Funktionalität müsste im Server dupliziert werden
  - Befragung von Saros-Entwicklern, Investigation im Code
- **Nicht-Kern-Abhängigkeitgraphen erstellen**
  - Welche Abhängigkeiten hindern am Verschieben in den Kern?
- **Abhängigkeiten entfernen**
  - Von außen nach innen
  - Verschiedene Refaktorisierungsstrategien (siehe nächste Folie)

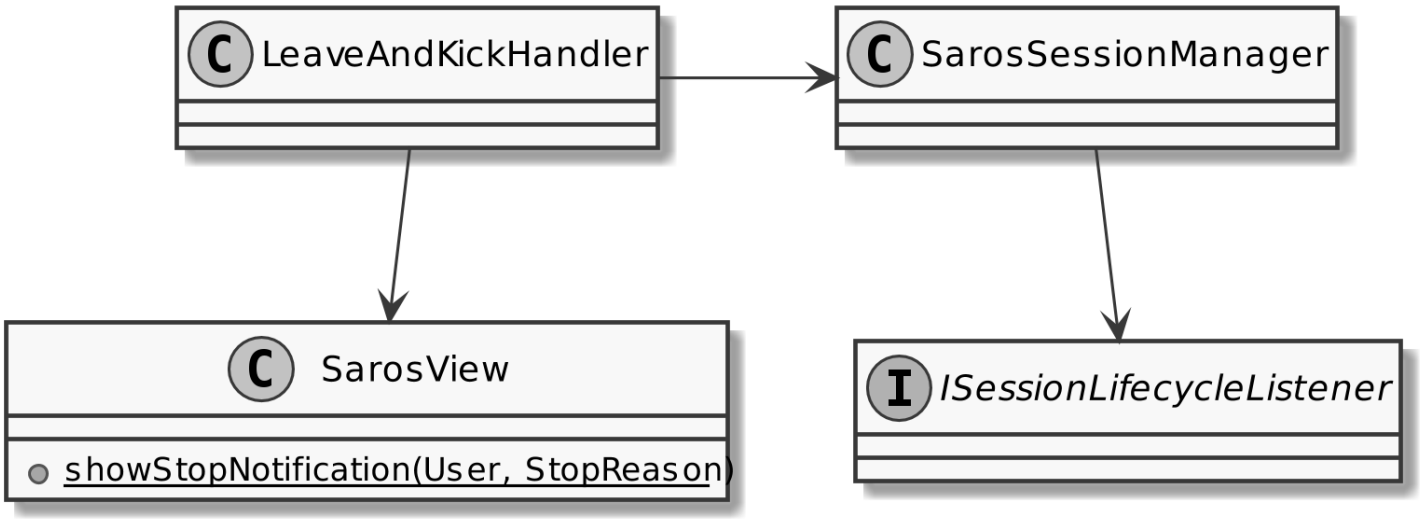
Kern-Erweiterung: **Abhängigkeitsgraph**



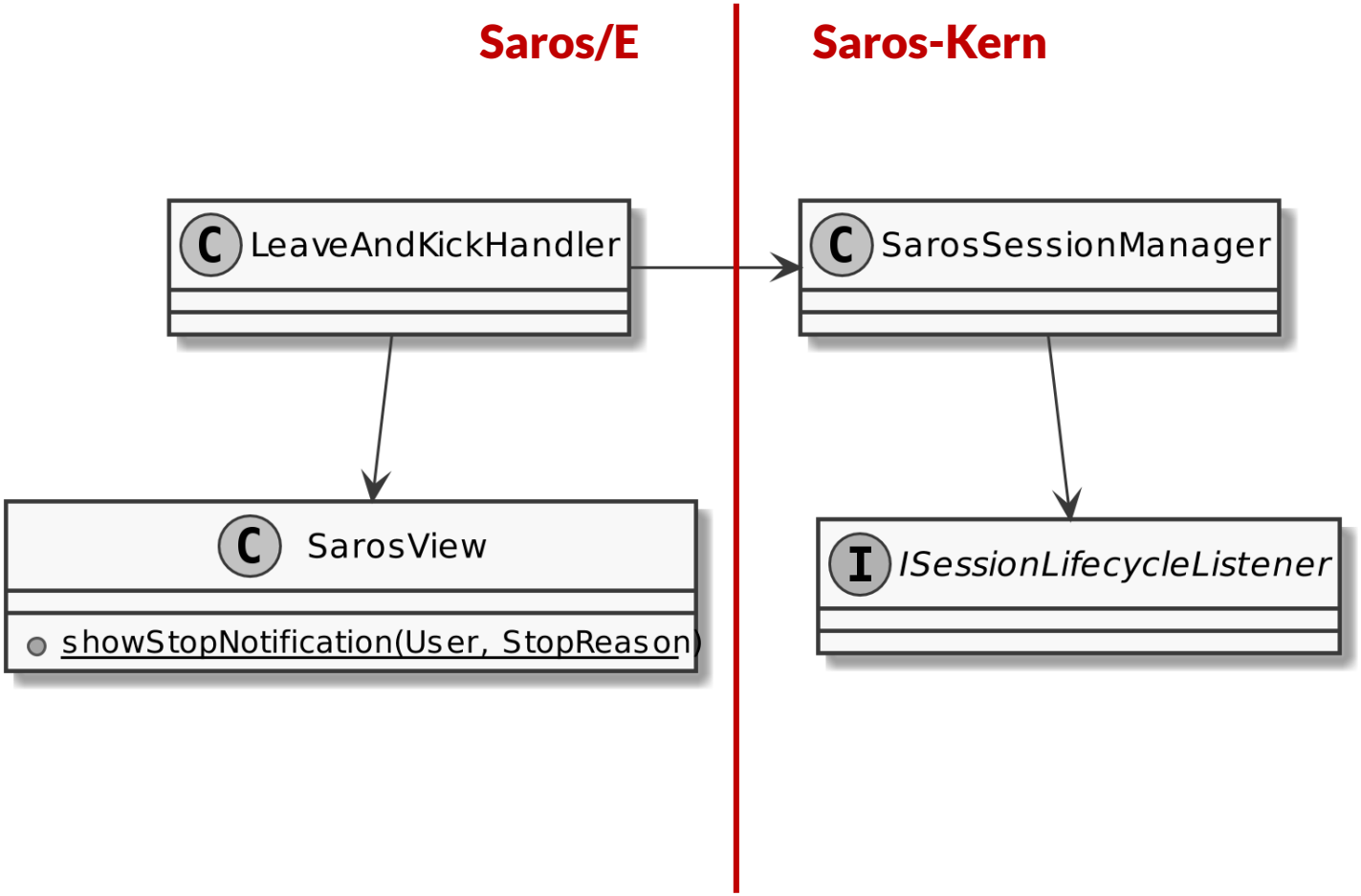
Kern-Erweiterung: **Refaktorisierungsstrategien**

- **Plattformtyp-Abhängigkeit → Kerntyp-Abhängigkeit**
  - Abhängende Klasse nutzt statt Eclipse-/IntelliJ-API ein Kern-Interface, dass in Saros/E und Saros/I „plattform“-spezifisch implementiert wird
- **Nicht-Kerntyp-Abhängigkeit → Kerntyp-Abhängigkeit**
  - Nicht-Kerntyp zur Implementation eines neuen Interfaces im Kern machen; abhängende Klasse das Interface nutzen lassen
  - Sinnvoll, wenn
- **Nicht-Kerntyp in den Kern verschieben**
  - Wenn alle problematischen Abhängigkeiten beseitigt

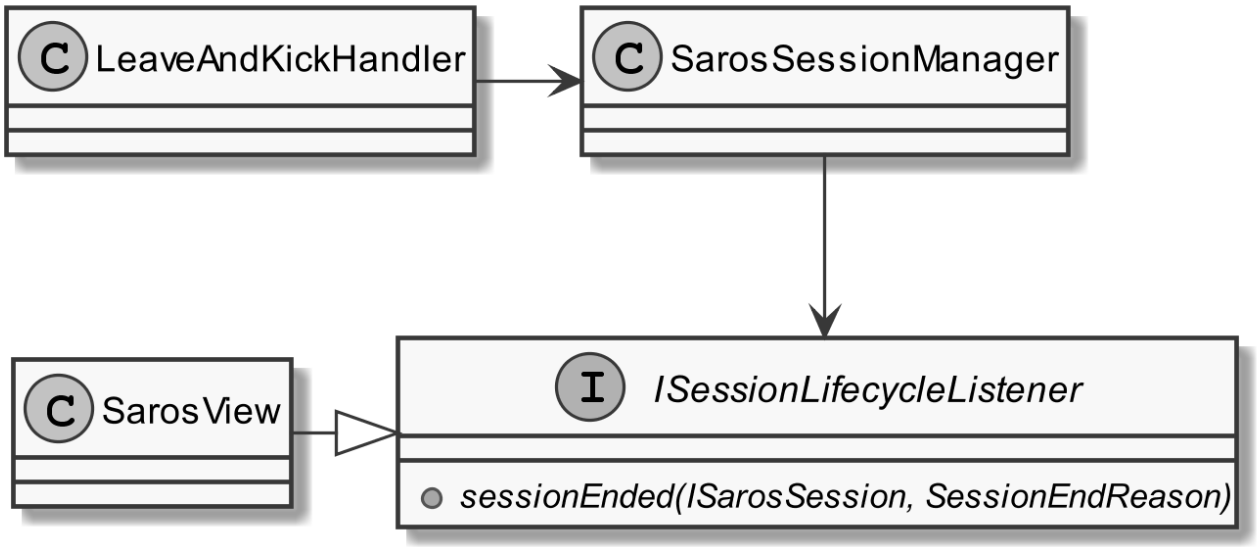
Kern-Erweiterung:
Beispielrefaktorisierung - Vorher



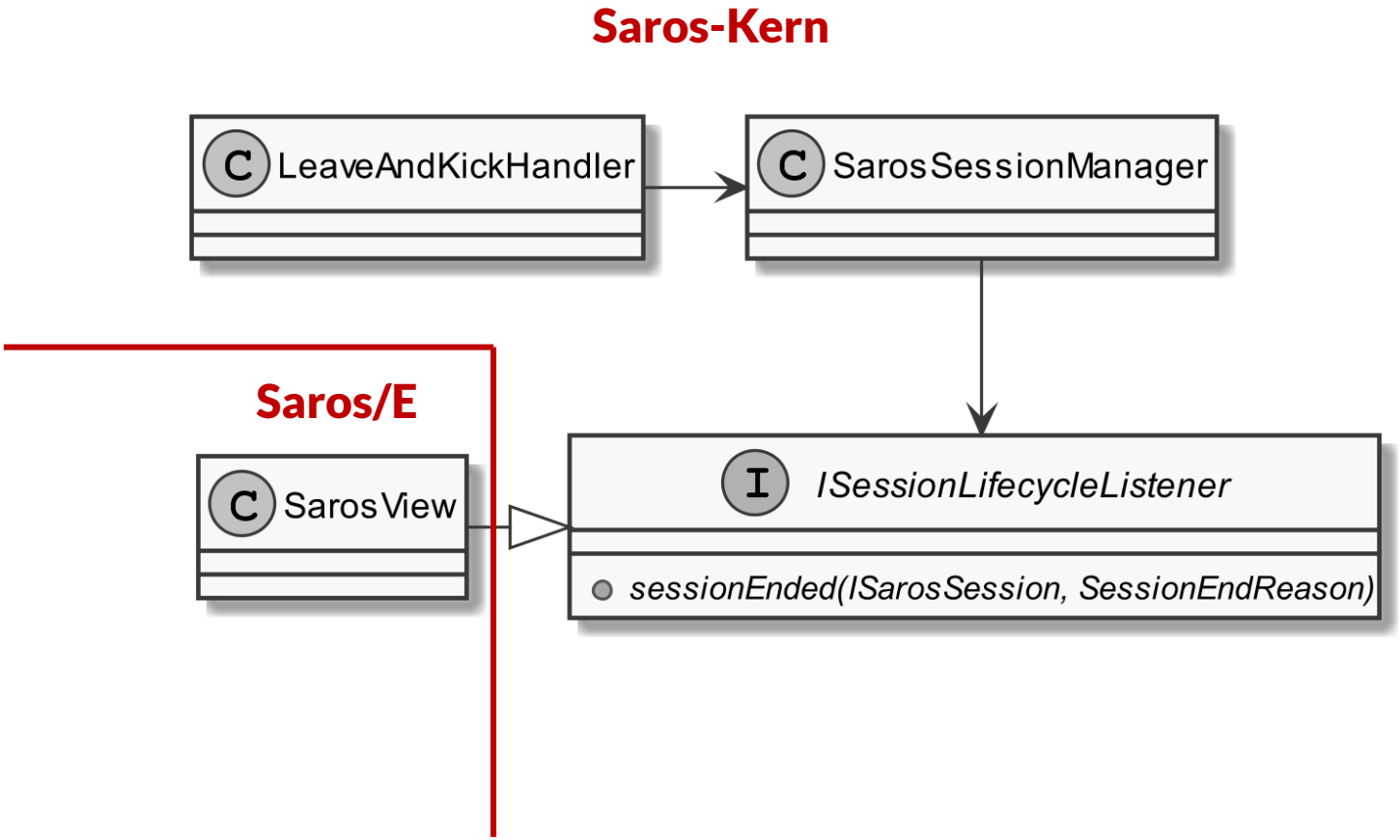
Kern-Erweiterung:
Beispielrefaktorisierung - Vorher



Kern-Erweiterung:
Beispielrefaktorisierung - Nachher



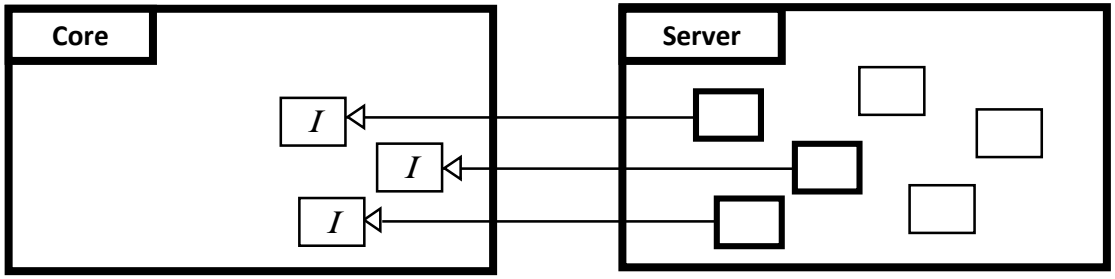
Kern-Erweiterung:
Beispielrefaktorisierung - Nachher



Kern-Erweiterung: **Status**

- Fast alle ermittelten essenziellen Klassen in den Kern migriert
  - **Project Negotiation** (`OutgoingProjectNegotiation`, `IncomingProjectNegotiation`)
  - **Sitzungsverwaltung** (`SarosSession`, `SarosSessionManager`, `LeaveAndKickHandler`)
  - Zahlreiche Abhängigkeiten
- 45 Patches (inkl. Cleanup und Dokumentationsergänzung)
- Alles bereits in den Saros-Hauptzweig intergiert

Implementation des Servers



- **Implementation von Kern-Interfaces**
  - Projekt- und Dateisystemmodell (`IWorkspace`, `IResource`, `IPath`, ...)
  - Einstelllungen (`IPreferencesStore`)
  - ...
- **Server-spezifische Komponenten**
  - Adaption des Server-Codes von Bussas
- **Bootstrapping-Code** (Main-Klasse usw.)

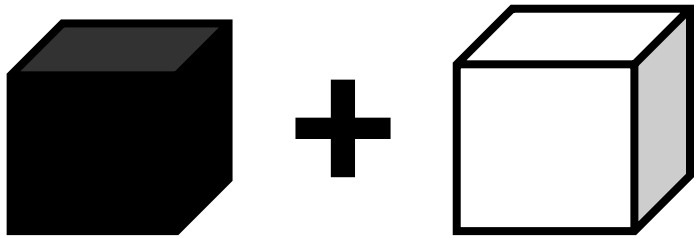
Implementation des Servers: **Entwurf**

- Unterstützt “Server Session Request”-Protokoll von Bussas
  - Aber ohne Unterscheidung zwischen “Create”- u. “Join”-Request (Beitritt wenn laufende Sitzung, neue Sitzung sonst)
- Kommandozeilenprogramm ohne IDE-Abhängigkeit
  - Nicht-interaktiv als Hintergrundprozess ausführbar
  - Konfiguration über Java System-Properties (`-Dkey=value`)
- Alle Entscheidungen automatisiert
  - Nimmt alle Anfragen (Kontaktliste, Server Session Request, Projekte von Nicht-Hosts) automatisch an
  - Konfiguration vorstellbar

Implementation des Servers: **Status**

- Funktionierende Implementation in **Change #2929**
- Einschränkungen
  - Nur eine Sitzung
  - Sitzung kann nicht beendet werden (aber wieder aufgenommen)
  - Kleinere Kinderkrankheiten (siehe Masterarbeit)
- Soll als Serie kleiner Patches in den Hauptzweig einziehen
  - Einige Patches bereits integriert

Evaluation der Server-Ressourcenverwaltung

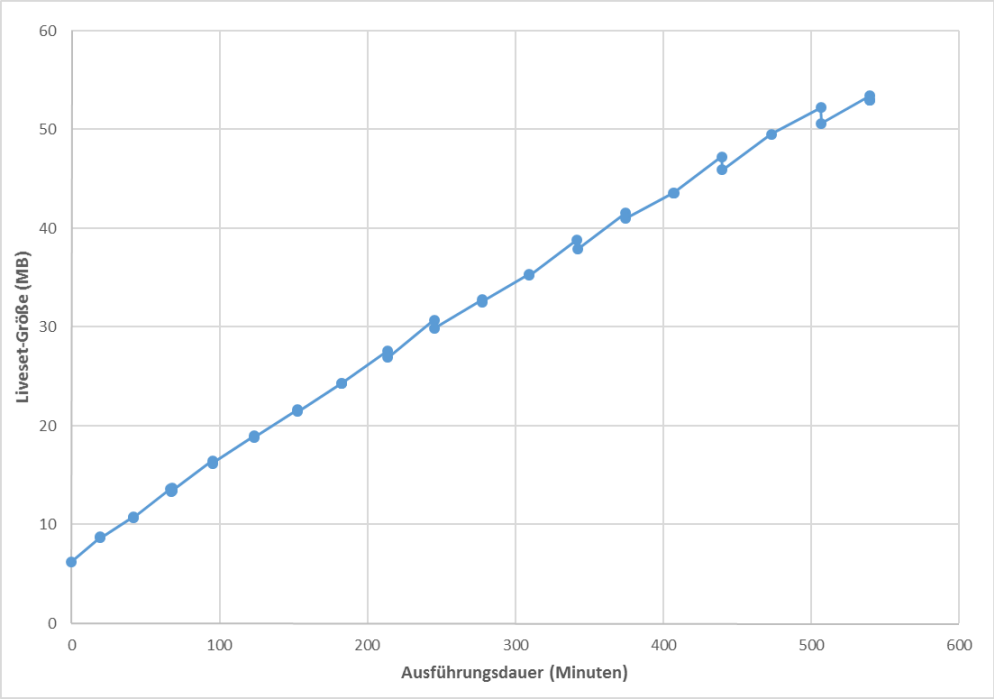


- **Frage:** Hat der Server oder der Kern Ressourcenlecks, die den Langzeitbetrieb gefährden?

- **Laufzeitanalyse** (Black Box)
  - Automatisierte Stresstests, Beobachtung des Hauptspeicherverbrauchs
  - Stresstests “speicherverbrauchsneutral” → steigender Verbrauch = mögliches Leck

- **Statische Analyse** (White Box)
  - Analyse der Codebasis mit FindBugs und PMD, um einfache Formen von Ressourcenlecks zu finden

Evaluation der Server-Ressourcenverwaltung: **Ergebnisse**



- Mit Laufzeitanalyse ein relevantes Speicherleck im Kern aufgespürt
  - Tritt auf, wenn ein Sitzungsteilnehmer kein Code bearbeitet (z.B. Server)
  - Ansätze zur Behebung in der Arbeit skizziert

Evaluation der Server-Ressourcenverwaltung: **Ergebnisse (2)**

- PMD / FindBugs haben keine Probleme gefunden
  - Aber hätten nur sehr einfach gestrickte Ressourcenlecks entdeckt
- **Kein Aufschluss über generelle Qualität der Ressourcenverwaltung**
  - Analyse nur punktuell, nicht umfassend und systematisch genug
  - Formales **Benutzungsmodell** würde Entwurf realistischer Lasttests erlauben

Zusammenfassung

Erreichtes

- Umsetzung eines Protokolls zum **Projektteilen als Nicht-Host**
- Erweiterung des Saros-Kerns um mehrere essenzielle Komponenten
- Ein einfacher **unabhängig lauffähiger Sitzungsserver**
- **Entdeckung eines Speicherlecks**

Offen

- Integration von Non-Host Project Sharing in den Hauptzweig
- Inkrementelle Überarbeitung und Integration des Server-Codes
- Lösung für mehrere Sitzungen pro Server
  - Idee: **Superserver**, der dynamisch Sitzungsserver erstellen / beenden kann (siehe Arbeit)

Danke fürs Zuhören

Fragen?