

Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Relokation und Dynamisches Binden in TinyOS

FG Betriebssysteme

Koblenz, 30. Juni – 1. Juli 2005

Pedro José Marrón, Matthias Gauger, Andreas
Lachenmann, Daniel Minder, Kurt Rothermel

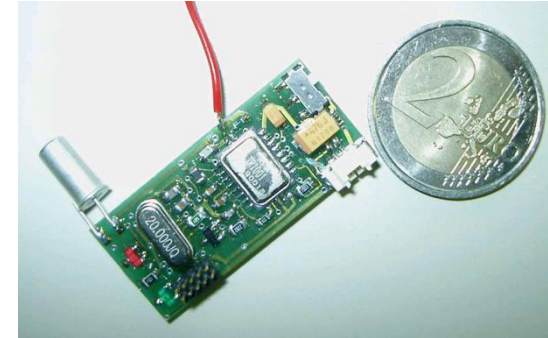
Gliederung

- Grundlagen
- Motivation und Zielsetzung des TinyCubus Projekts
- FlexCup
 - Grundkonzept
 - Ablauf
 - Optimierungen
- Evaluation
- Zusammenfassung und Ausblick



Was sind Sensornetze?

- Sammlungen von kleinen Geräten, die:
 - mit **Sensoren** ausgestattet sind
 - **Ad-Hoc** miteinander kommunizieren
 - **Daten** austauschen und bearbeiten
 - **begrenzte Ressourcen** haben
 - meistens **stationär** sind
- Herausforderungen
 - Lange Betriebsdauern erforderlich (Mehrere Jahre!)
 - Ressourcenbeschränktheit
 - Manuelle Eingriffe (meist) nicht möglich



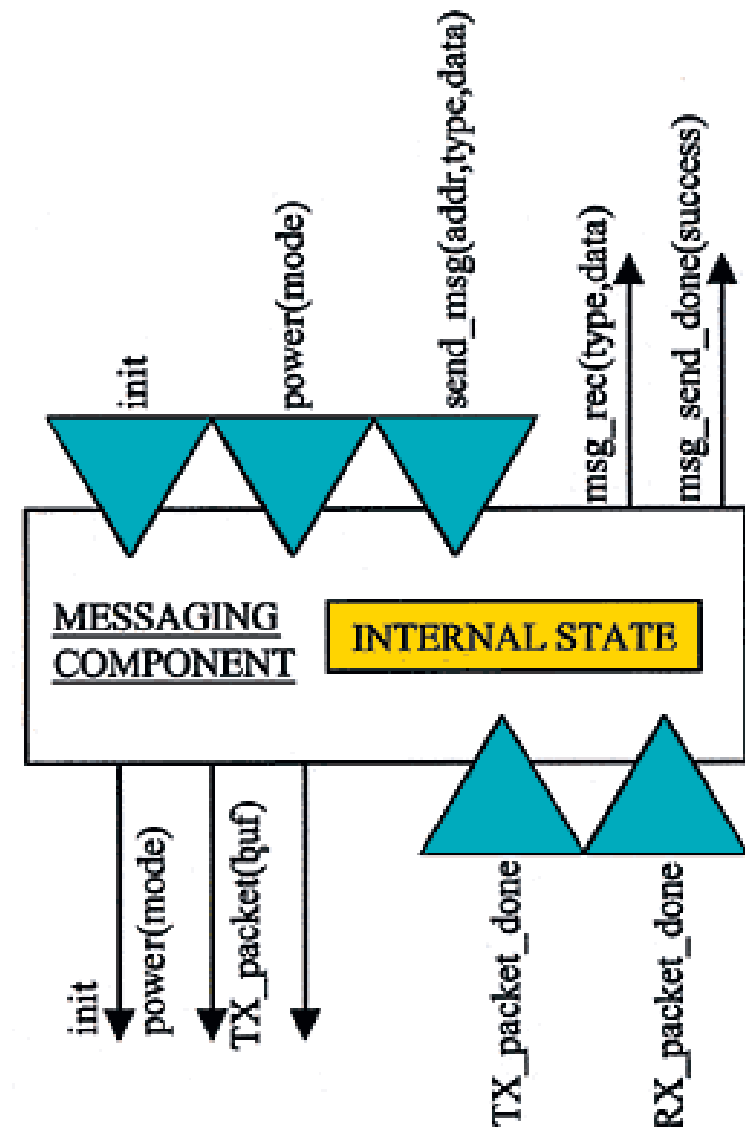
Mica2 Sensorknoten

- Prozessor: Atmega 128L
 - AVR-Architektur (RISC-Variante)
 - 128 KB Programmspeicher, 4KB SRAM, 4KB interner EEPROM
 - Programmspeicher im laufenden Betrieb beschreibbar
- 528 Kilobyte externer Flash
- Funkschnittstelle:
 - Proprietäres Übertragungsverfahren im ISM-Band
 - 19,2 kbps Datenrate
 - Reichweite im Freien max. 150 – 300m



Das TinyOS Projekt

- **Ziel:** Entwicklung eines minimalistischen „Betriebssystems“ für Sensornetze
- **Eigenschaften:**
 - Event-driven Architektur
 - nesC als Programmiersprache
 - Unterstützt das Komponentenmodell von TinyOS
 - Komponenteninterface:
 - provides, uses, signals and handles
- Messaging Component Beispiel



Merkmale von TinyOS

- Betriebssystemeigenschaften:
 - Single-shared stack
 - Keinen richtigen Kernel, Prozessverwaltung, Speicherverwaltung oder virtuelle Speicher
 - Einfaches FIFO 2-level Scheduling
 - Multithreading nur zwischen Events und Anwendungen
- Standardkomponenten:
 - Timer und Uhreninterfaces
 - ADC: Generic Sensing Interface
 - RFM, UART: Funk- und Serielle-Kommunikationskomponenten
 - Power Management Interface



Anwendungsbeispiele

- **Habitat Monitoring**
 - Great Duck Island (GDI) System
 - Hogthorb -- Sow heat period monitoring
- **Umweltbeobachtung und Forecasting-Systeme**
 - ALERT -- National Weather Service
 - Floodnet – Überwachung von Flüssen in UK
- **Gesundheitsanwendungen**
 - Care in the Community -- UK
 - UbiCare -- UK
- **Militäranwendungen**
 - WINS -- Surveillance and exploration
 - Odyssey -- Underwater surveillance



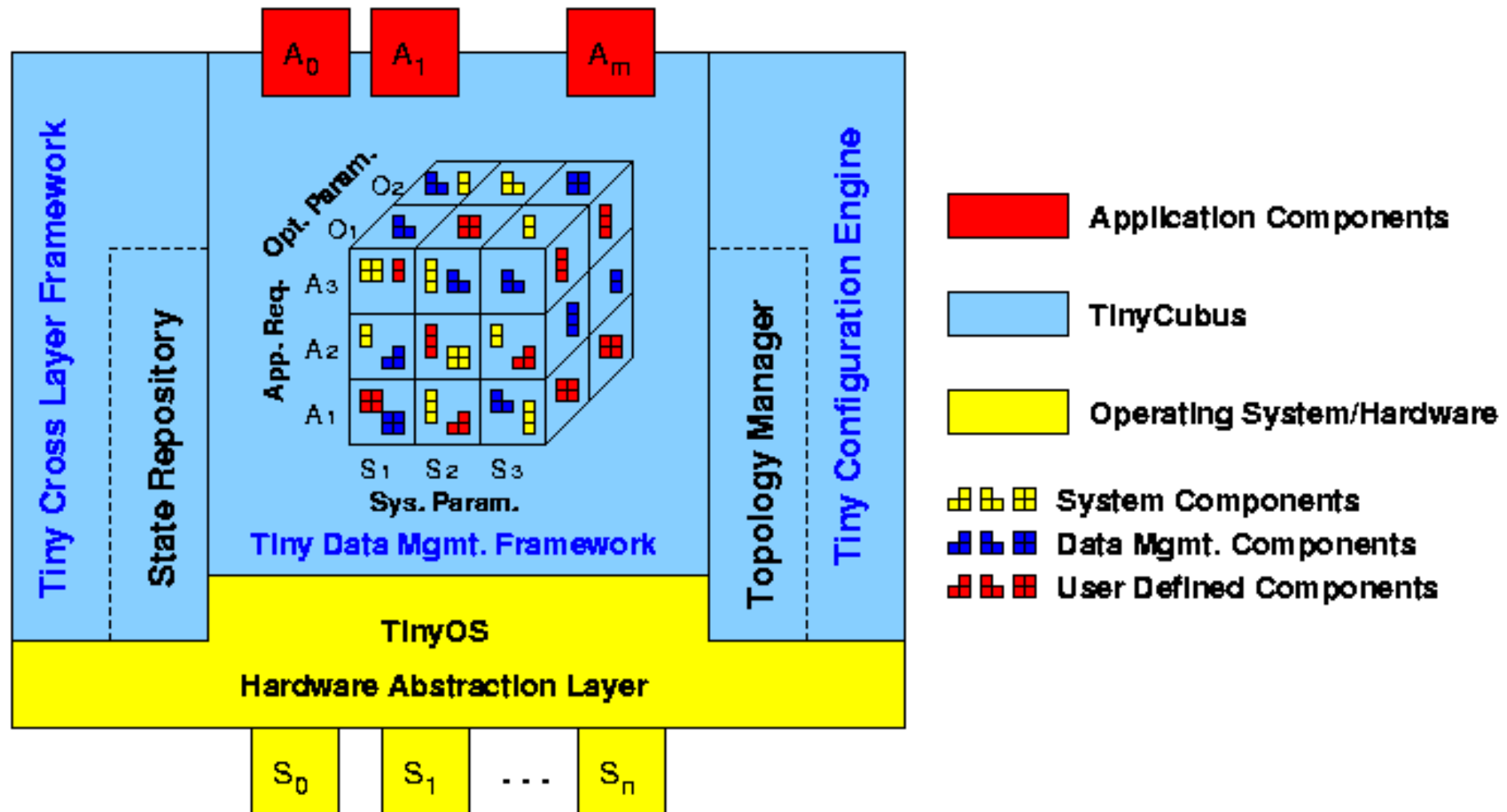
- **Gebäude Beowachtung**
 - Structure Health Monitoring System -- US, Canada
 - Sustainable Bridges -- EU
- **Intelligente Verkehrssysteme**
 - Safe Traffic -- Schweden
 - Vehicular Networks (CarTalk 2000) -- EU
- **Smart Room Umgebungen**
 - Aware Home -- Georgia Institute of Technology
 - Sense-R-Us -- Universität Stuttgart
- . . . und viele mehr



- **Herausforderung:** Steigende Komplexität der Sensornetzanwendungen
- **Ziel:** Unterstützung für Sensornetzanwendungen durch flexible und adaptierbare Systemsoftware
- **Vorgehensweise:**
 - Implementierung auf der Basis von TinyOS
 - Definition allgemeiner Frameworks für Adaption
 - Bereitstellung von standard Komponenten
 - Systemkomponenten
 - Datenverwaltungs- und Anfragebearbeitungskomponenten

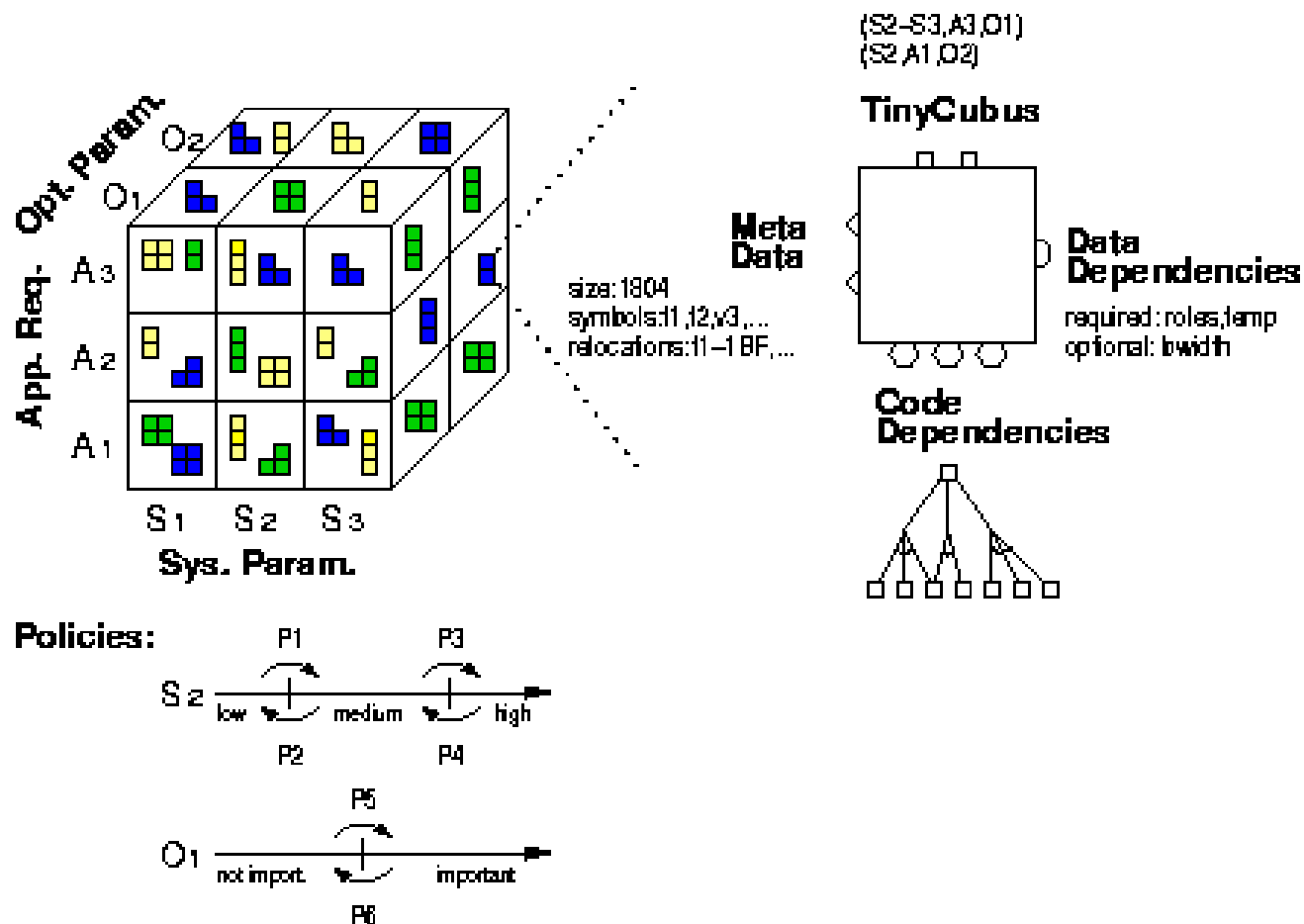


TinyCubus Architektur



Tiny Data Management Framework

- **Ziel:** Adaptionsstrategien und Bereitstellung von Datenverwaltungs-komponenten



Tiny Cross-Layer Framework

- **Ziel:** Unterstützung der Parametrisierung von Komponenten und Verwaltung der Datenabhängigkeiten

Name	Type	Publishers	Subscribers	Data
roles	l_{roles}	(system)	req:C3	$n1=\{r1\}$
comp	l_{comp}	(system)	(system)	$n1=\{C1,C2,C7\}$
pol	l_{pol}	(system)	(system)	$n1=\{S1,(10,27,35)\}$
temp	float	C1,C5	req:C4,C5	$n3=24.01$
bwidth	int	C2	req:C5,opt:C3	$\{n1,n3\}=42$



Tiny Configuration Engine

- **Ziel:** Unterstützung für die Rekonfiguration und Installation von Komponenten
- **Drei Teile:**
 - Topology Manager
 - Rollenverwaltungsalgorithmen
 - Flexible Code Update Algorithmus (FlexCup)
- **Anforderungen an FlexCup:**
 - Dynamischer Austausch einzelner Codeteile
 - Unterstützung für autonome Anpassungen einzelner Knoten



Problemstellung

- Gelegentliche Softwareupdates notwendig
 - Korrektur von Fehlern
 - Anpassung an neue Aufgaben oder Umweltbedingungen
- Aber: Feste Bindung zwischen Anwendung und Betriebssystem
 - Kein Update einzelner Anwendungsteile möglich
 - Vollständiger Codeaustausch sehr energieaufwändig
 - Fehlende Flexibilität zur Adaption

Idee

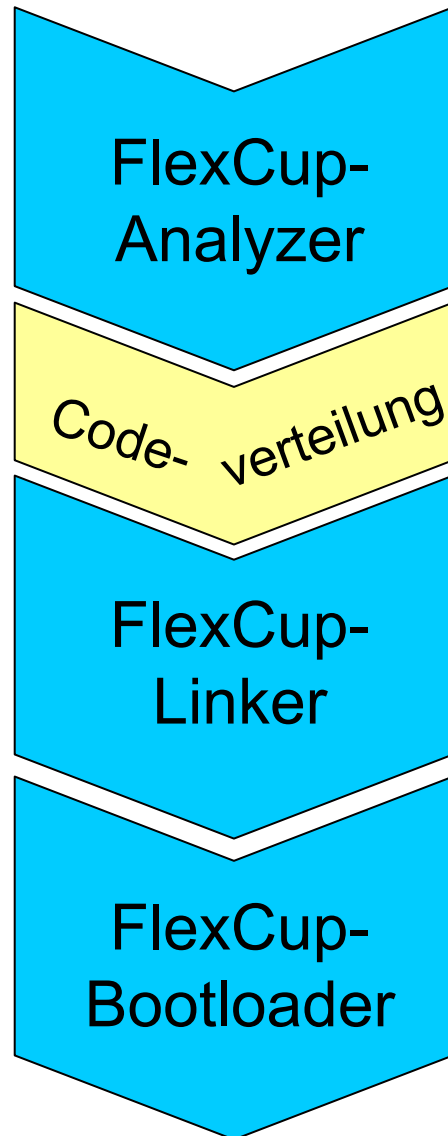
- Austausch einzelner Komponenten
- Dynamische Integration auf den Sensorknoten



FlexCup – Flexible Code Updates

- Grundkonzept: Dynamischer Austausch von Komponenten
 - Erhalten und Ausnutzen der Komponentenstruktur von nesC
 - Bei Codeupdates: Nur geänderte Komponenten übertragen
 - Integration der neuen Komponente auf den Sensorknoten
 - Austausch des alten Programmcodes
 - Aktualisieren der Referenzen
 - FlexCup nutzt Metadaten
 - Allgemeine Programminformationen
 - Symboltabelle mit Funktions- und Objektsymbolen
 - Relocationtabelle
- ↳ Benötigen Speicherplatz und Übertragungsvolumen!

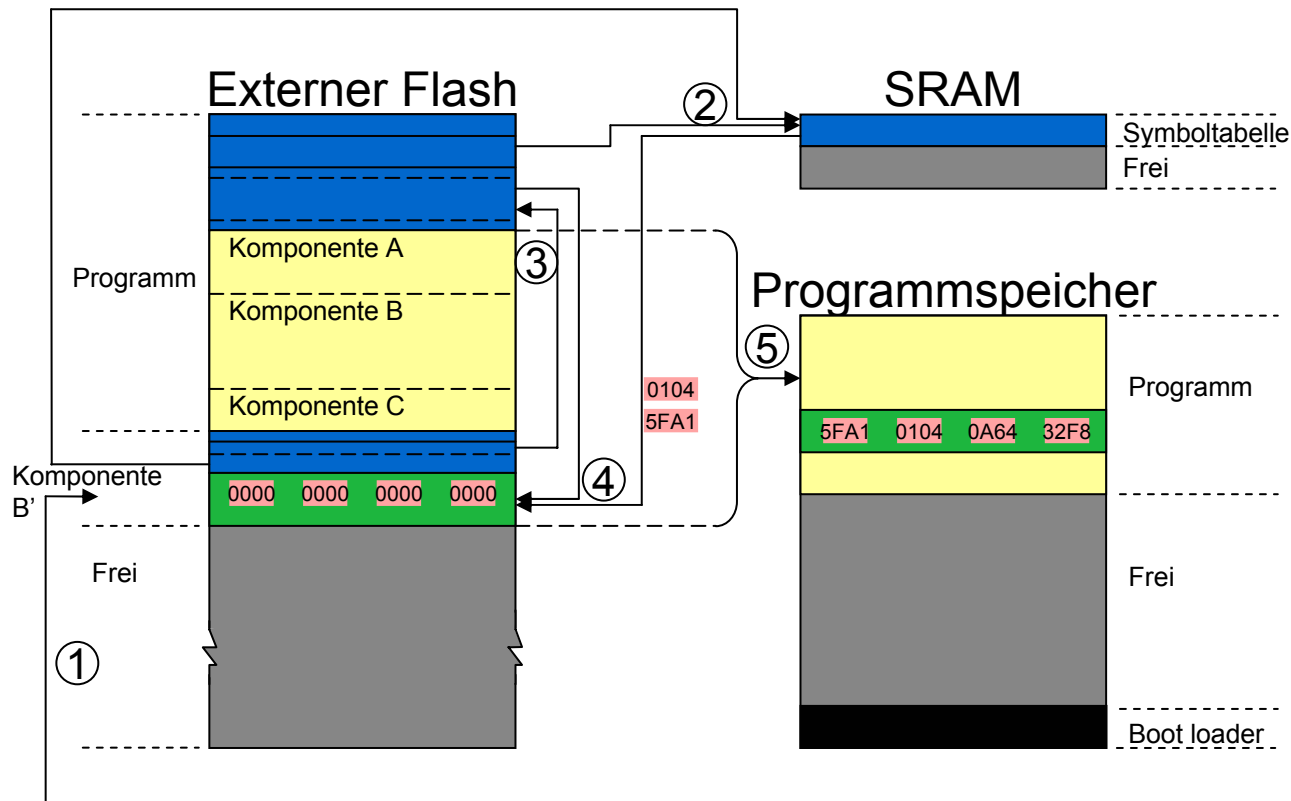




- Aufteilung der Anwendung in Komponenten
- Auswahl zu übertragender Komponenten
- Erstellen benötigter Metadaten
- Integration der neuen Komponente
- Aktualisieren der Metadaten
- Anpassen der Referenzen
- Übertragung des neuen Programms in den Programmspeicher
- Neustart des Systems



Ablauf auf den Sensorknoten

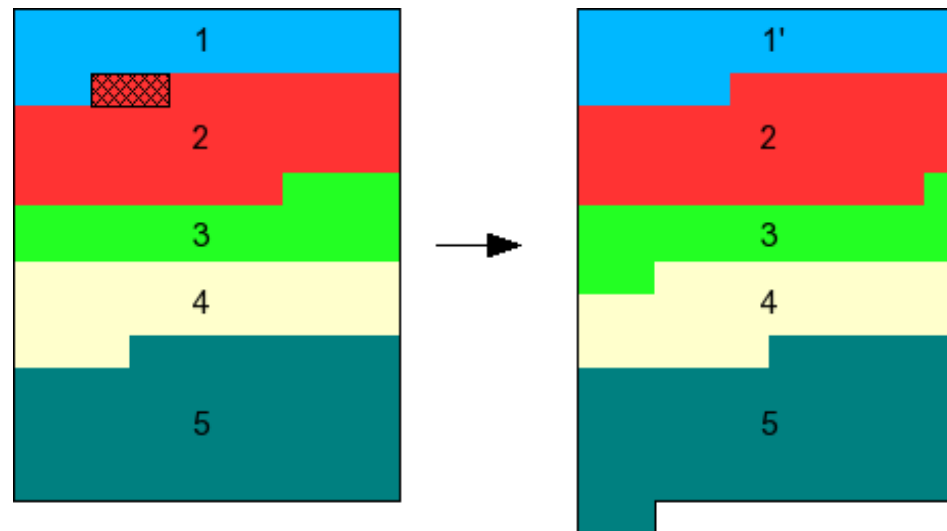


- 1) Empfang der neuen Komponente
- 2) Aktualisieren der Symboltabelle
- 3) Austausch der Relocationtabelle
- 4) Anpassen der Referenzen
- 5) Übertragen in den Programmspeicher



Optimierungen

- **Problem:** Zugriffe auf den externen Flash besonders teuer
- ↳ **Ziel:** Anzahl der Zugriffe minimieren
- Mehrere Ansätze implementiert:
 - Bewusster Einsatz der Puffer des Flashbausteins
 - Pufferung der Symboltabelle im RAM der Sensorknoten
 - Pufferbereiche („Slop Space“) zwischen den Programmkomponenten



Verwandte Arbeiten

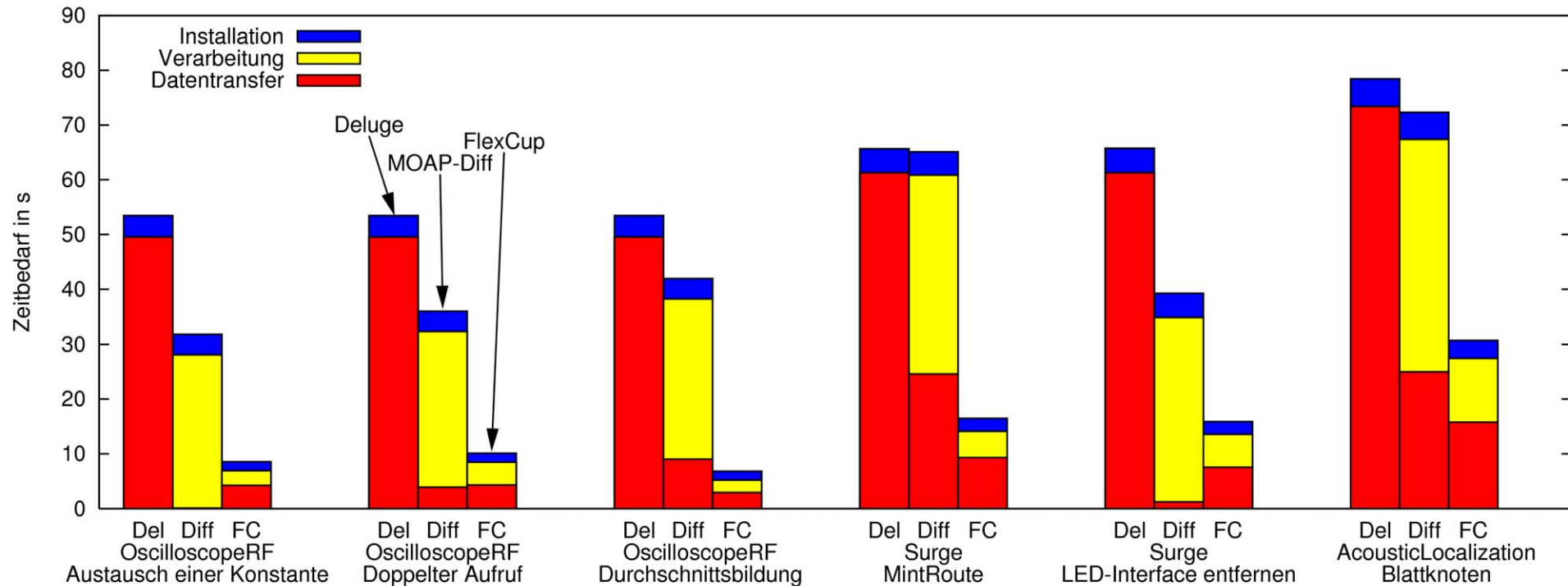
- Komplettaustausch (z.B. Deluge)
 - Übertragung des kompletten Codeimages an die Sensorknoten
 - + Einfache Implementierung
 - Große Datenmengen
 - Ineffizient bei kleinen Änderungen
- Differenzansätze (z.B. MOAP-Diff)
 - Übertragung der Differenz zur Vorversion
 - + Nur geänderte Codeabschnitte müssen übertragen werden
 - Erfordert Wissen der Basisstation – keine Adaption einzelner Knoten
 - Kleine Änderung – große Auswirkungen?
- Betriebssystem- und Middlewarelösungen (z.B. SOS, Contiki)



- Vergleich mit zwei alternativen Ansätzen: Deluge und MOAP-Diff
- Praktische Untersuchungen anhand von
 - Drei Beispielanwendungen unterschiedlicher Komplexität
 - Sechs Änderungen am Programmcode der Anwendungen
- Annahmen
 - Keine Kollisionen und Übertragungsfehler, kein Protokolloverhead
 - Knoten empfängt Update einmal und leitet es einmal weiter
- Simulationen mit *atemu*
 - Emuliert Atmel-Mikrocontroller und Mica2-Hardware
 - Erlaubt präzise Ermittlung des Zeit- und Energiebedarfs



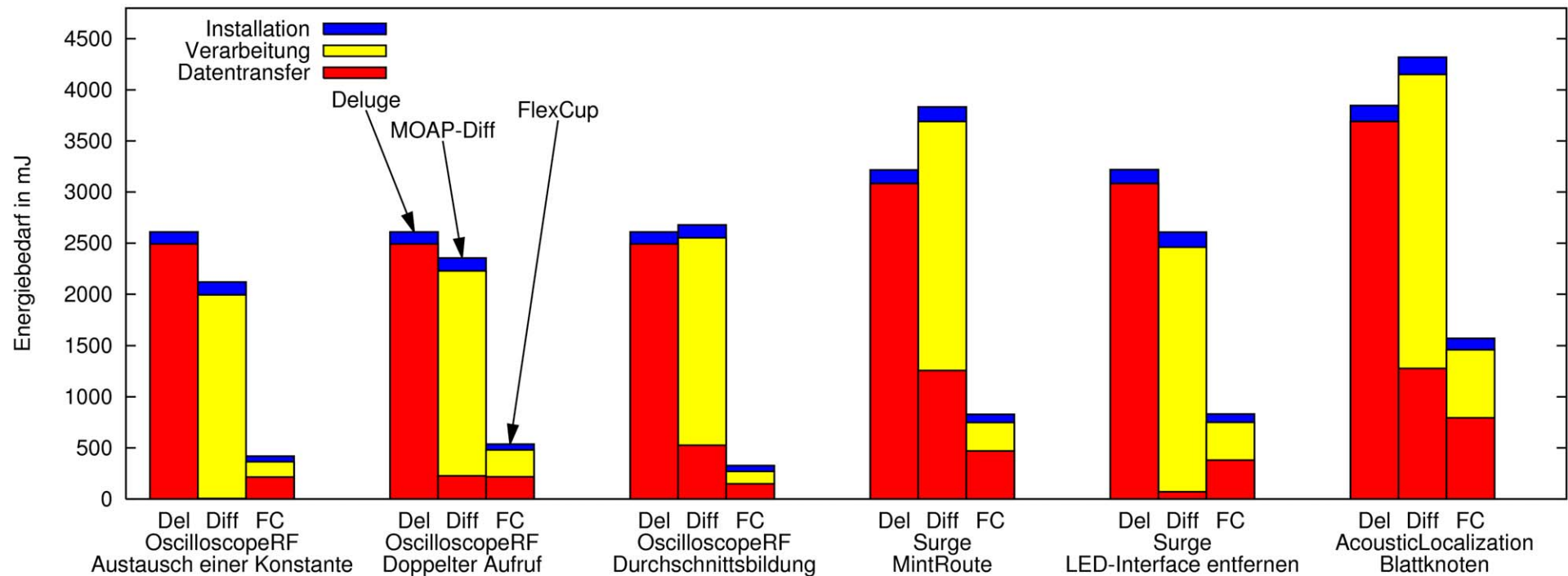
Evaluation: Zeitbedarf



- Einsparungen beim Datentransfer durch Verarbeitungsaufwand
- FlexCup mindestens 2,5 mal schneller als Deluge und 2,3 mal als MOAP-Diff



Evaluation: Energiebedarf



- Einfluss des Verarbeitungsaufwands geringer als bei MOAP-Diff
- Max. 41 % der Energie von Deluge und 37 % von MOAP-Diff



Zusammenfassung

- FlexCup realisiert den dynamischen Austausch von Komponenten
 - Bietet die für TinyCubus benötigte Flexibilität
 - Einsparungen bei der Datenübertragung
- Sehr gute Performance im Vergleich zu verwandten Ansätzen
 - Bis zu achtmal schneller
 - Bis zu 87 % Einsparungen beim Energieaufwand
 - Aber: Einsatz von FlexCup benötigt Platz im externen Flash



- Implementierung für weitere Systemplattformen
- FlexCup Diff
 - Kombiniere FlexCup mit der Idee der Differenzansätze
 - Erste Versuche durchaus vielversprechend
 - Aber: Verlust der Unabhängigkeit von der Basisstation
- Integration von FlexCup in TinyCubus
 - Codeverteilungsalgorithmus
 - Adaptionskomponente
- + Jede Menge sonstige interessante Probleme



Vielen Dank für die Aufmerksamkeit!



Gibt es Fragen?



IPVS

Research Group

“Distributed Systems”

25

Universität Stuttgart

IPVS

Backup-Folien



Komplexität der Beispielanwendungen

	OscilloscopeRF	Surge	AcousticLocalization
Größe unverteilt (in Bytes)	11784	17096	24272
Größe zerteilt (in Bytes)	14116	20420	28020
Anzahl der nesC-Komponenten	39	53	69
Anzahl der Binärkomponenten	6	10	15



Übertragene Datenvolumina (in Bytes)

Anwendung	Updatetyp	Deluge	MOAP-Diff	FlexCup		
				Meta	Code	Gesamt
OscilloscopeRF	Austausch einer Konstante	23142	11	799	1198	1997
	Doppelter Aufruf	23142	1230	801	1202	2003
	Durchschnittsbildung	23142	2835	537	886	1423
Surge	MintRoute	28652	7684	1056	3258	4314
	LED-Interface entfernen	28652	375	1355	2142	3497
Acoustic-Localization	Blattknoten	34162	7802	2565	4773	7338



Platzverbrauch im externen Flash (in Bytes)

Anwendung	Updatetyp	Deluge	MOAP-Diff	FlexCup
OscilloscopeRF	Austausch einer Konstante	23142	28538	37337
	Doppelter Aufruf	23142	28542	37343
	Durchschnittsbildung	23142	28608	36743
Surge	MintRoute	28652	33440	43561
	LED-Interface entfernen	28652	34272	42744
Acoustic-Localization	Blattknoten	34162	40156	58014



- **Herausforderung:** Steigende Komplexität der Sensornetzanwendungen
- **Ziel:** Unterstützung für Sensornetzanwendungen durch flexible und adaptierbare Systemsoftware
- Drei Teile
 - Tiny Data Management Framework: **Adaption**
 - Tiny Cross-Layer Framework: **Cross-Layer Interaktionen**
 - Tiny Configuration Engine: **Codeverteilung und –installation**
- Anforderungen an FlexCup:
 - Dynamischer Austausch einzelner Codeteile
 - Unterstützung für autonome Anpassungen einzelner Knoten



- TinyOS – das Betriebssystem
 - Quasi-Standard für Sensornetzanwendungen
 - Menge von Systemkomponenten + Scheduler
 - Keine Trennung zwischen Betriebssystem und Anwendung
- nesC – die Programmiersprache
 - Erweiterung der C-Programmiersprache
 - Unterstützt das Komponentenmodell von TinyOS
 - Compiler verbindet System- und Anwendungskomponenten zu monolithischem Programm

