OSE - 2006

Betriebssystemtechnik

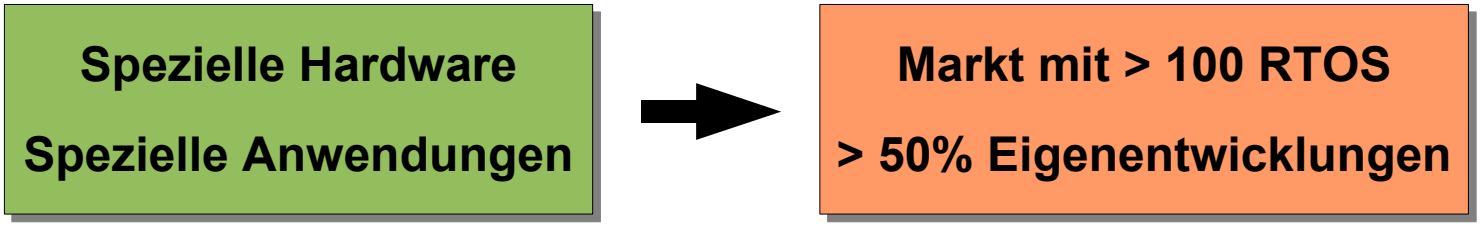
Olaf Spinczyk
Julio Sincero

Lehrstuhl für Informatik IV
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

{os, sincero}@cs.fau.de

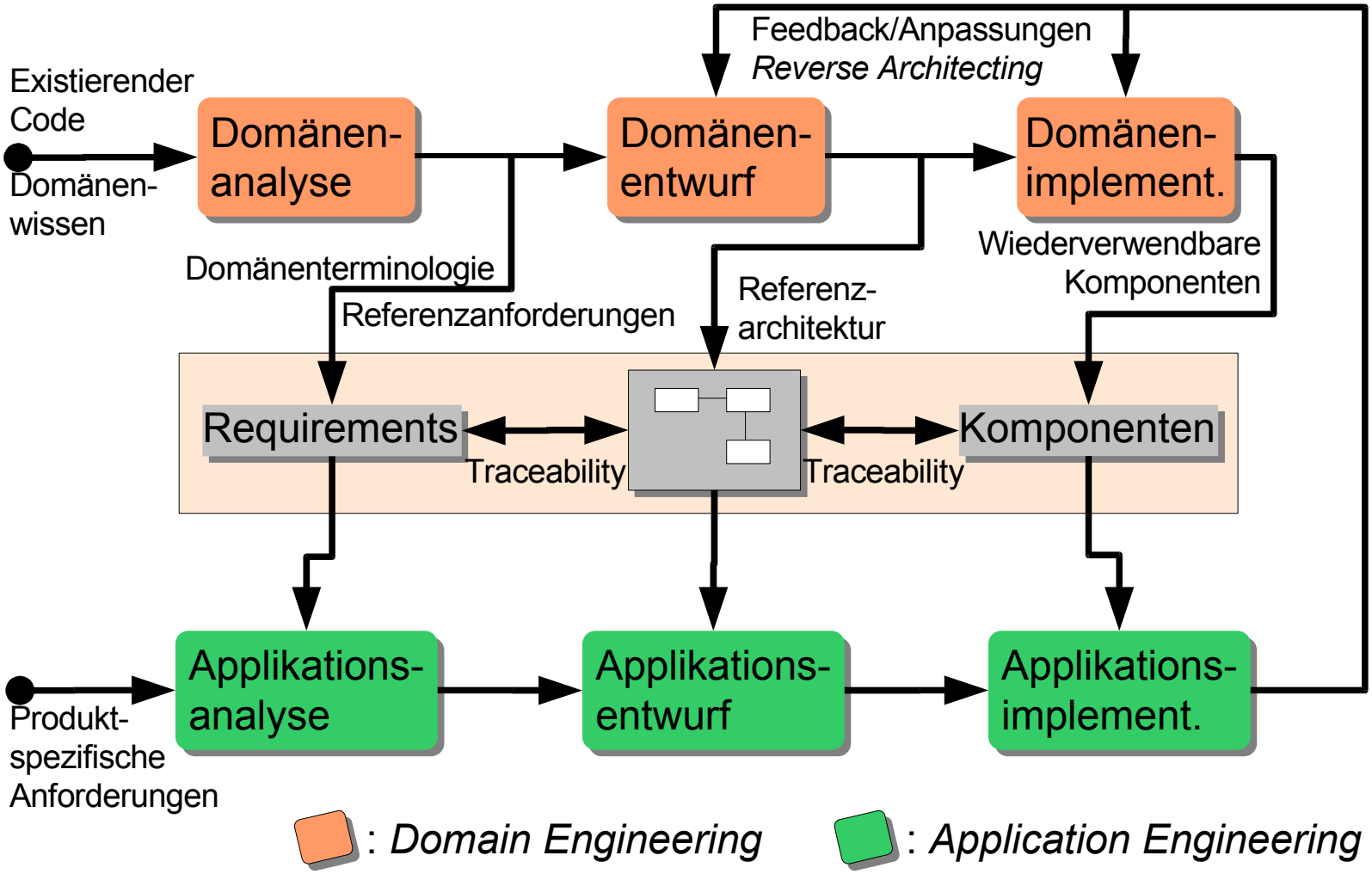
Eingebettete BS Entwicklung heute

Betriebssysteme für eingebettete Systeme



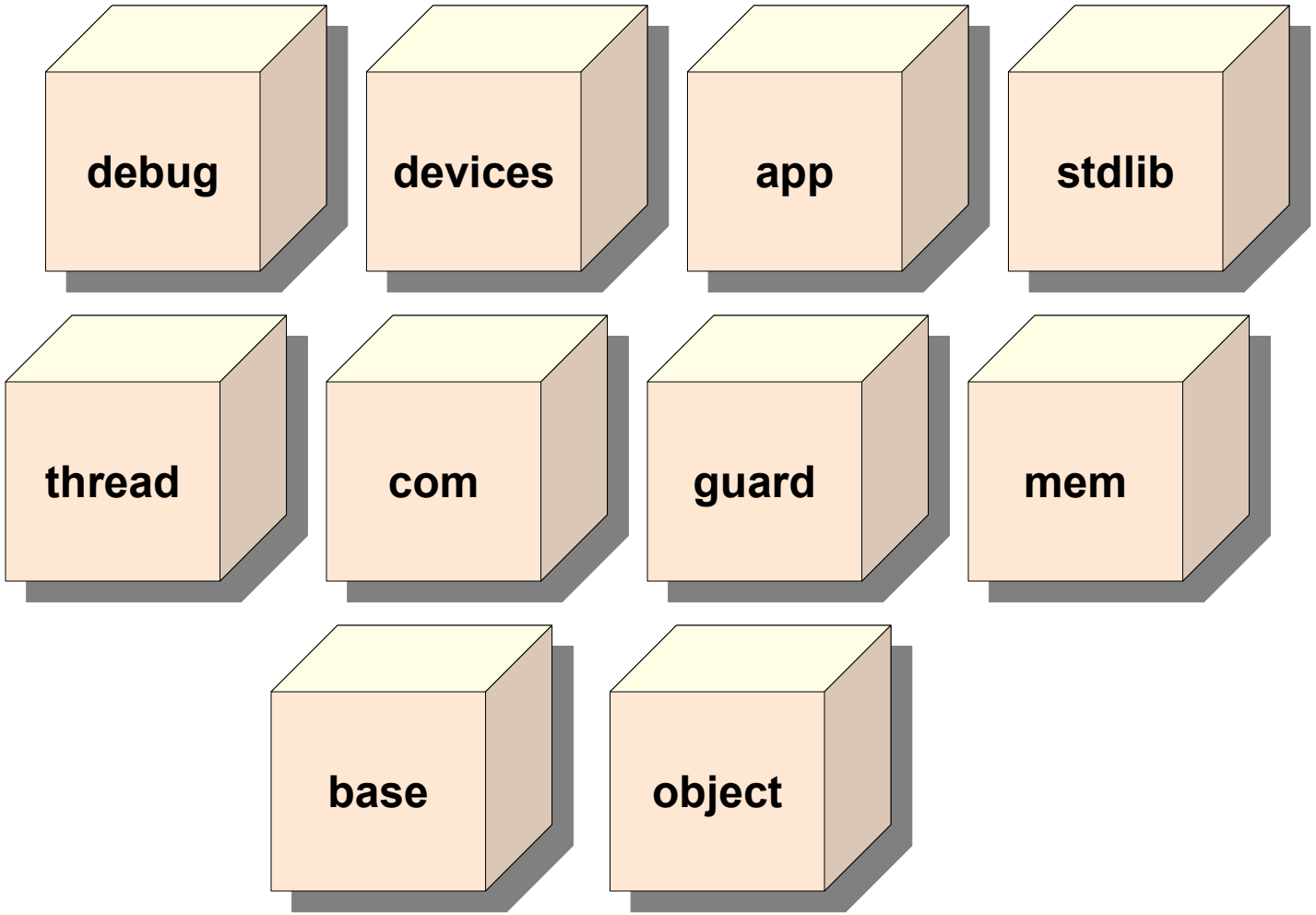
- „das Rad wird neu erfunden“
  - auch die selben Fehler werden wiederholt
- oftmals bietet **ein** BS Hersteller **mehrere** Systeme an
  - mit getrennter Code-Basis
  - getrieben durch die speziellen Anforderungen seiner Kunden

Software-Produktlinienentwicklung

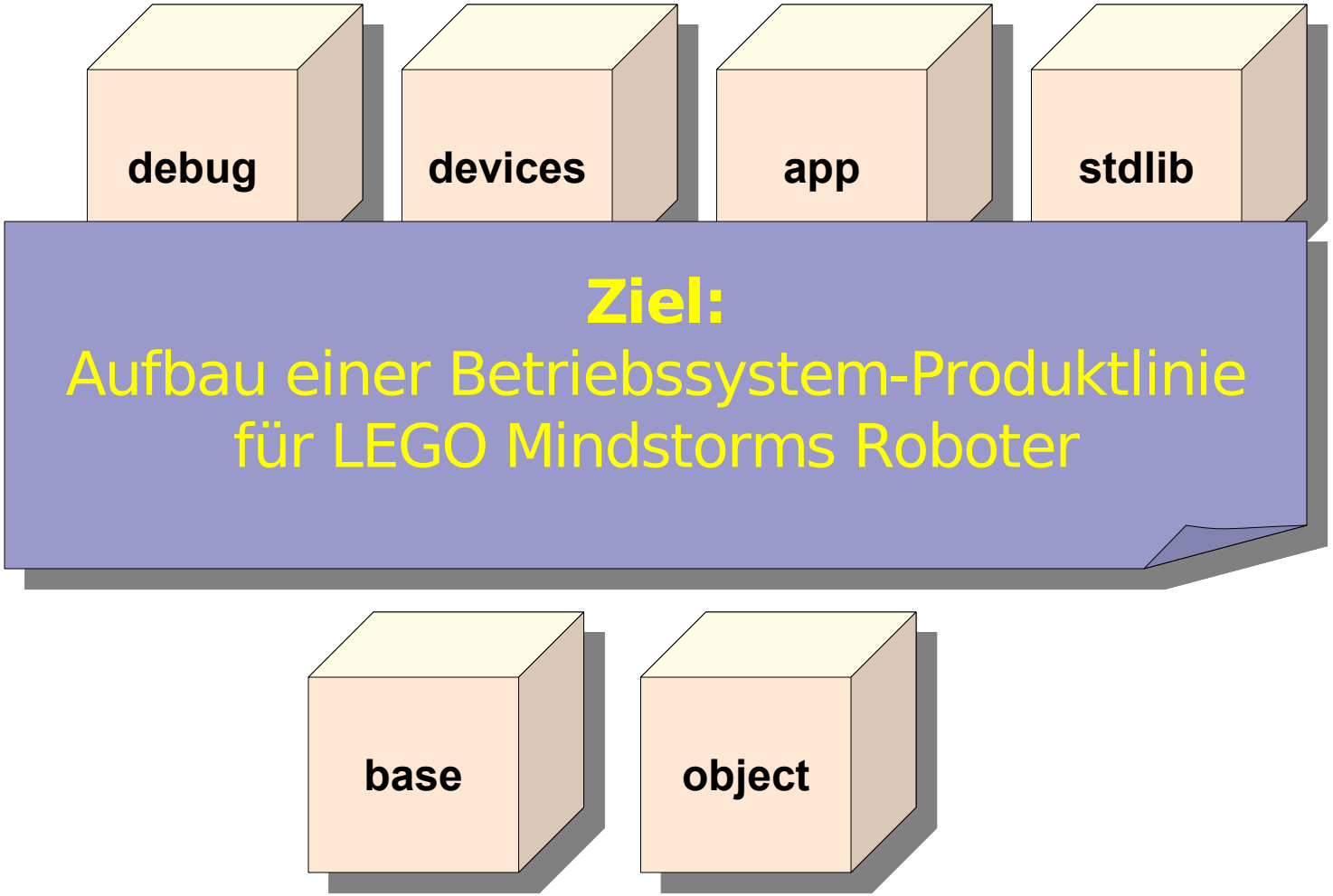


Referenzprozess für die Software-Produktlinienentwicklung [1]

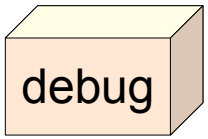
Arbeitsaufteilung



Arbeitsaufteilung

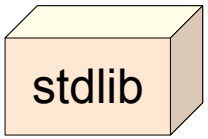


Debugging



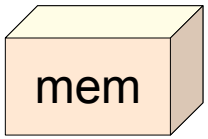
- Highlights:
  - DebugStream
    - `debug::out << „fehler!“ << debug::yell() << debug::blink(3) << debug:: endl;`
  - DebugToolbox (in Form von Aspekten)
- DynamicAssertion
  - Infrastruktur/Makefile webt nicht in den Tests
- MemoryChecker
  - Zeitmangel

Standard Library



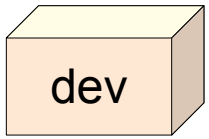
- Container
  - STL konforme Klassen
  - Vector, String, List, Queue, PriorityQueue, Stack, Set, Map
  - ChainList, ChainQueue
- Streams
  - Output (Display, Infrarot)
  - Input (Infrarot)
- Konfigurierbarkeit
  - Dynamischer / statischer Speicher
  - Output Alignment, Negative Zahlen
  - Thread-safety
  - Wenige Abhängigkeiten zu anderen Subsystemen

Memory Management



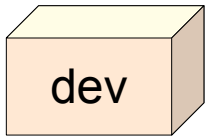
- Strategien
  - Welche Strategien können überhaupt genutzt werden?
  - Welcher Heap benutzt welche Strategie?
  - Geplant: Mehrere hintereinander geschaltete Strategien pro Heap, (z.B. MemPool mit 4 Bytes, dann MemPool mit 8 Bytes, u.s.w.)
- Callbacks
  - Was passiert, wenn nur noch wenig Speicher zur Verfügung steht?
  - Was passiert, wenn kein Speicher mehr allokiert werden kann?
- Statistiken
  - Wie viele Bytes sind allokiert?
  - Wie effizient ist die Strategie im Moment?
  - Wieviel ist noch frei?
- Anderes
  - Sind die Zugriffe Thread-synchronisiert?
  - Ist der Speicher initialisiert?

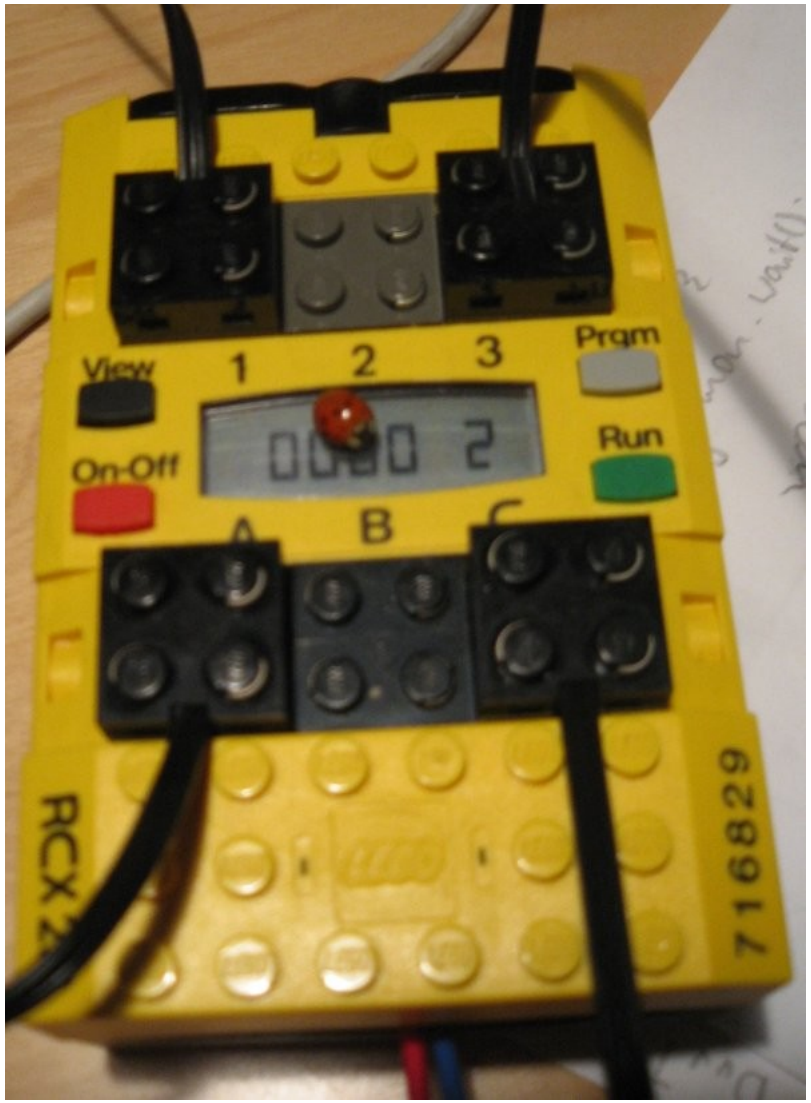
Devices



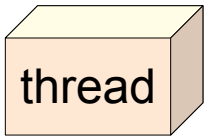
- konfigurierbare Unterstützung der Geräte:
  - Timer
  - Display
  - Motor
  - LED
  - Buttons (intern)
  - Batteriesensor
  - Buttons (extern)
  - Lichtsensor
  - Rotationssensor
  - Temperatursensor
- pro Gerät konfigurierbare Features:
  - im Thread-Betrieb:
    - Treiber läuft in AppThread: Monitore werden per Aspekt eingewoben
    - Treiber läuft in eigenem DriverThread
  - Callbacks
    - Callbacks werden über Aspekte mit virtuellen Pointcuts eingewoben
    - globaler InstanceManager verwaltet Callbacks für mehrere Instanzen der gleichen Klasse (z.B. für mehrere Sensoren)

Devices



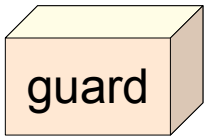


Thread

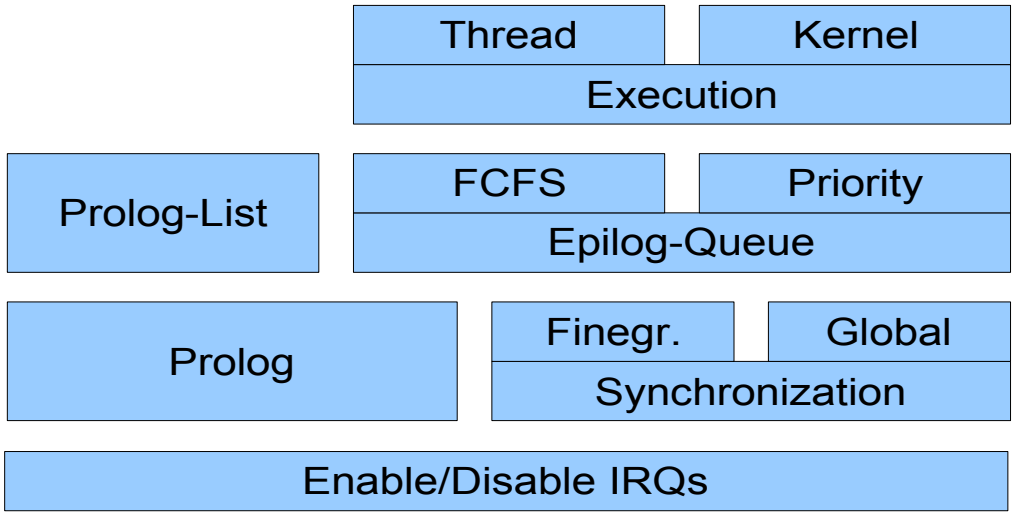


- Online und Offline Scheduler
- 3 Scheduling Algorithmen auswaehlbar
  - FCFS
  - Multilevel (frei waehlbare Anzahl an Ebenen)
  - RoundRobin
- Synchronisationsprimitiven Mutex, Semaphore und Monitor
- Preemption ist in eigenstaendiger Klasse implementiert
  (im Gegensatz zu OOStubs)

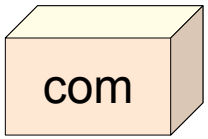
Guard



- Dynamische Prolog-Listen
- Rekonfigurierbares Interruptmodell
  - Harte/Weiche Synchronisation
  - Globales/Feingranulares Locking
  - Epilogausführung in Thread oder Kernel
- Keine Änderungen an „Benutzer“-Code nötig

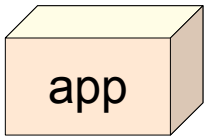


Inter-Process Com.



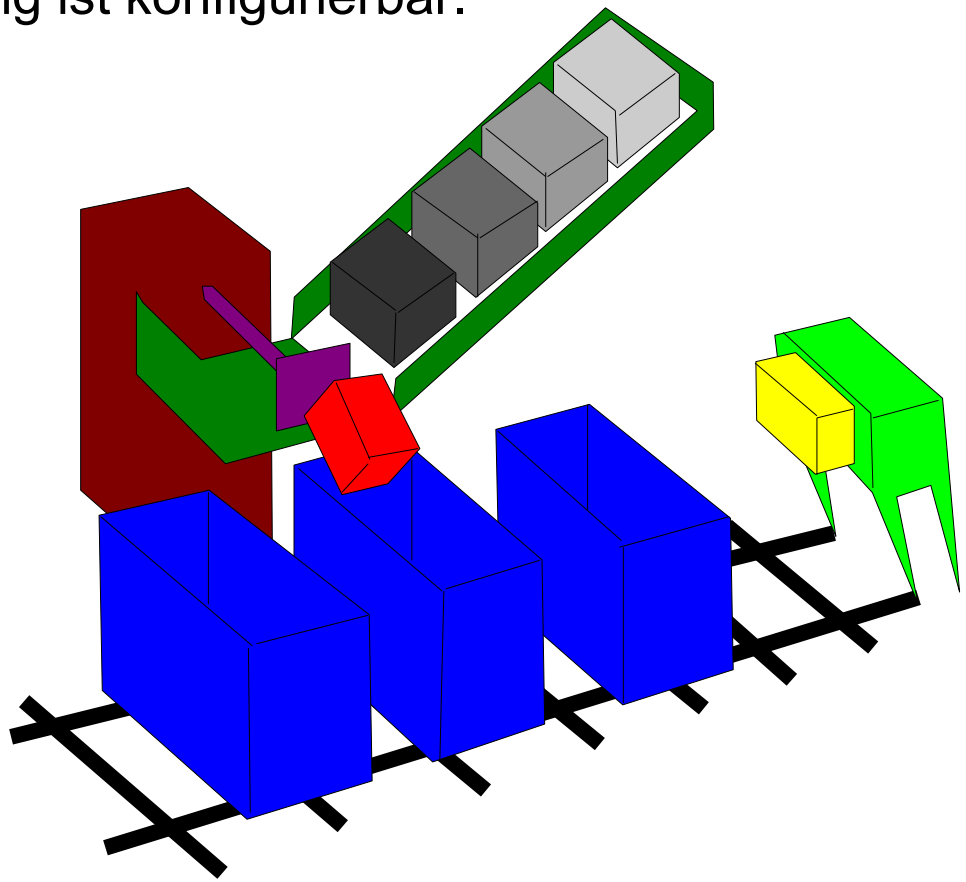
- 4 Strategien
  - 1:1 Sockets
  - 1:n Pipes
  - n:1 MessageQ
  - n:m SharedMem
- Asynchron oder Synchron
- Alle über „Registrierungs“-Singletons
- Infrarotport
  - Eigenes Kommunikationssystem (Zeitmangel)
  - Beruht auf MessageQ und Dispatcher

Applikation

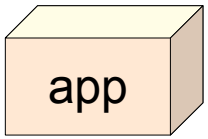


- Einsatzgebiet: Sortiermaschine

Legosteine werden in mehrere Container einsortiert. Sortierverfahren und Reporting ist konfigurierbar.

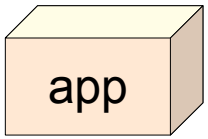


Applikation



- Statisch konfigurierbar (via P::V):
  - Grundlegende Eigenschaften
    - Containeranzahl
    - Geschwindigkeit
    - Sortierreihenfolge (Gleichverteilung, Vordefinierte Liste, Grauwertsensor, Eingabe via Buttons, Ferngesteuert über Infrarot)
  - Querschneidende Belange
    - Report-Generierung
    - dynamischer Puffer
    - Multithreading
    - Debugausgaben
- Realisiert über:
  - Pure::Variants Attribute und Flags
  - Klassen-Aliase
  - Aspekte

Applikation



- Größenvergleich verschiedener Applikationen der Produktlinie „Sortiermaschine“

	text	data	bss	gesamt
EQ-Distribution:	5122	191	2081	7394
Greyvalue:	5336	191	2081	7608
Predefined-List:	5364	191	2081	7636
+ div Reporting:	5476	191	2081	7748
Buttons:	5600	191	2081	7872
+ div Reporting:	5784	191	2085	8056
+ Heap, etc.:	6330	195	2363	8888

- Gute Skalierung:
  - Komponentenmodell beinhaltet Abhängigkeiten zum BS
  - Die benötigten Eigenschaften des BS (z.B. Threading, Synchr., ..) werden automatisch ausgewählt, nicht benoetigte Klassen sind nicht im Binary vorhanden => „Maßschneiderung“