

Linux Automation Konferenz 2005

Analyse von
Performanceverbesserungen
für ein
Baukastensystem zur Konstruktion von
Regelkreisen

Lehrstuhl für Software Engineering

Universität Hannover

Dominic Hillenbrand

Gliederung

- Umfeld & Anforderungen
- Ziel
- Legacy System Architecture
- Aufrufmechanismen
- Codegenerator
- Ausblick

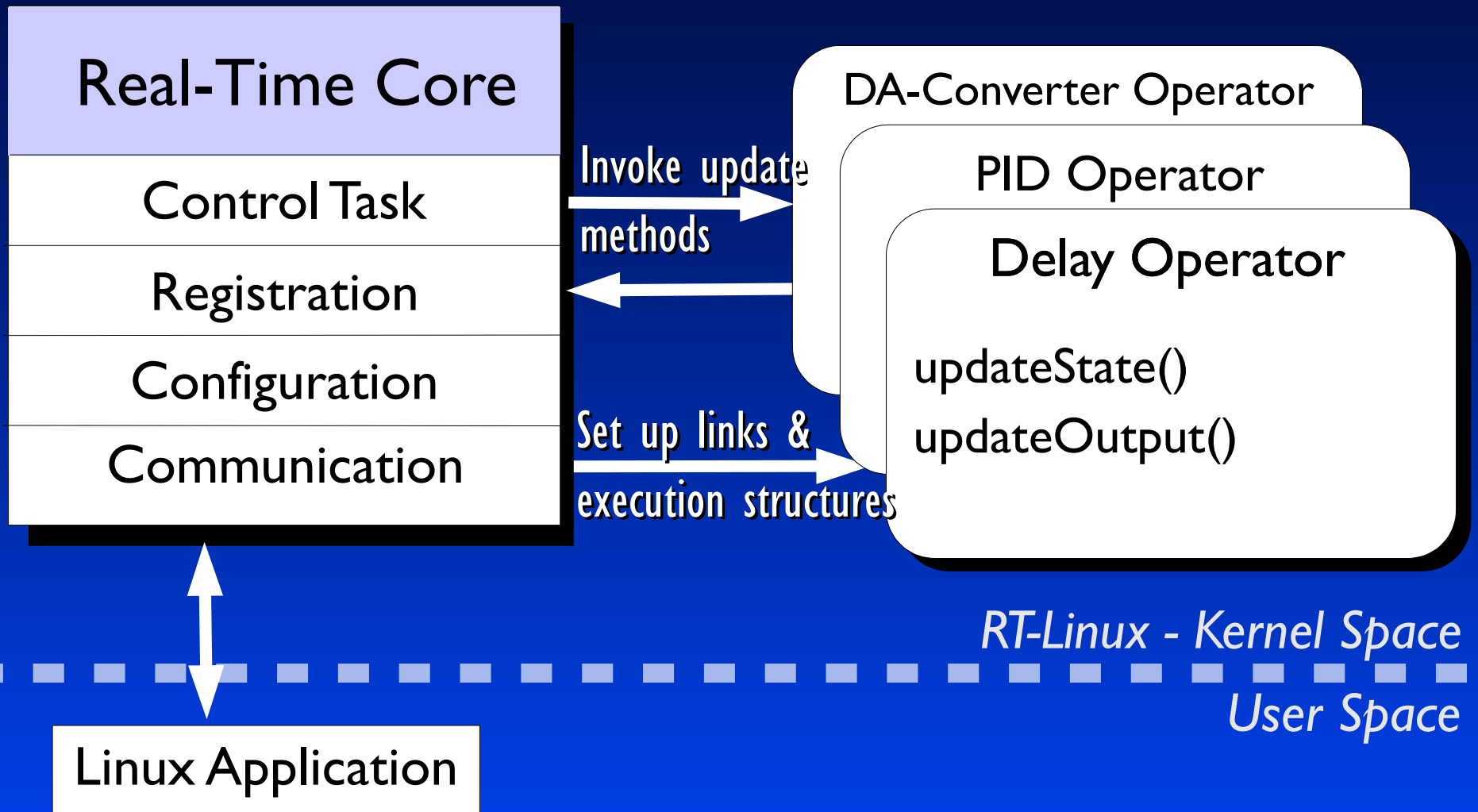
Umfeld & Anforderungen

- Reglerentwurf (HIL)
- Hohe Rechenleistung & Reglerfrequenz
 - Pentium > 1GHz
- Viele Ein- und Ausgänge (DAQ-Karte)
 - National Instruments NI-6035E
- Leistungsfähige Entwicklungsumgebung
 - Am Institut entwickelte Software
 - RT-Linux & KDE (KDevelop, ...)


Ziel

- Erhöhung der maximalen Regelfrequenz
 - Architektur
 - Regler:
 - Aufrufmechanismen
 - Quellcodeoptimierung

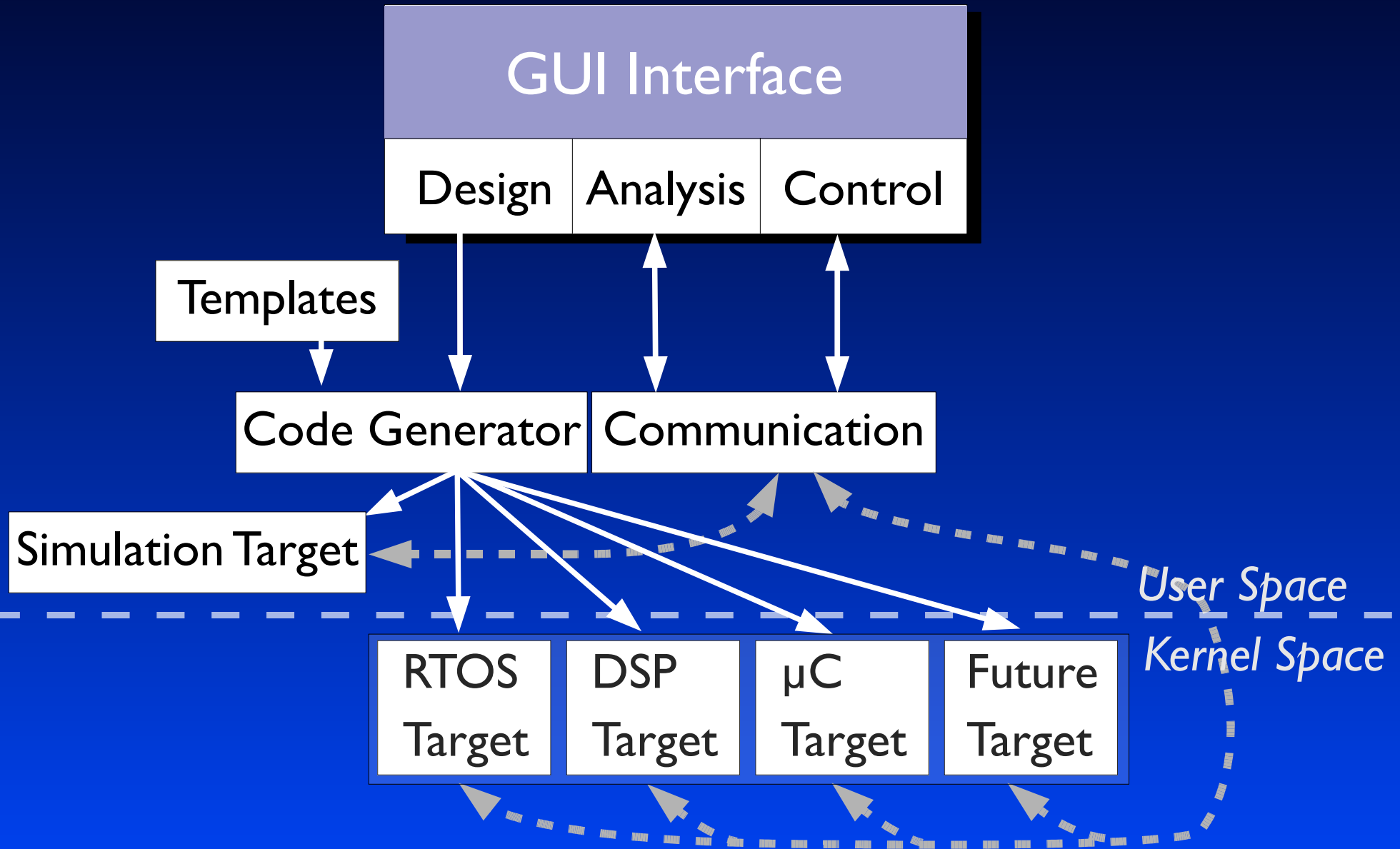
Legacy System Architecture 1/2



Legacy System Architecture 2/2

- 1:1 Mapping C++ Reglerbaustein zu Kernelmodul
- **Vorteile:**
 - dynamisches Laden & Entladen von Reglerbausteinen
 - dynamisches “Verdrahten” der Reglerbausteine
- **Nachteile:**
 - “zu langsam”
 - Programmierfehler  (Kernel)
 - An (RT)-Linux gebunden
 - C++ im Kernel

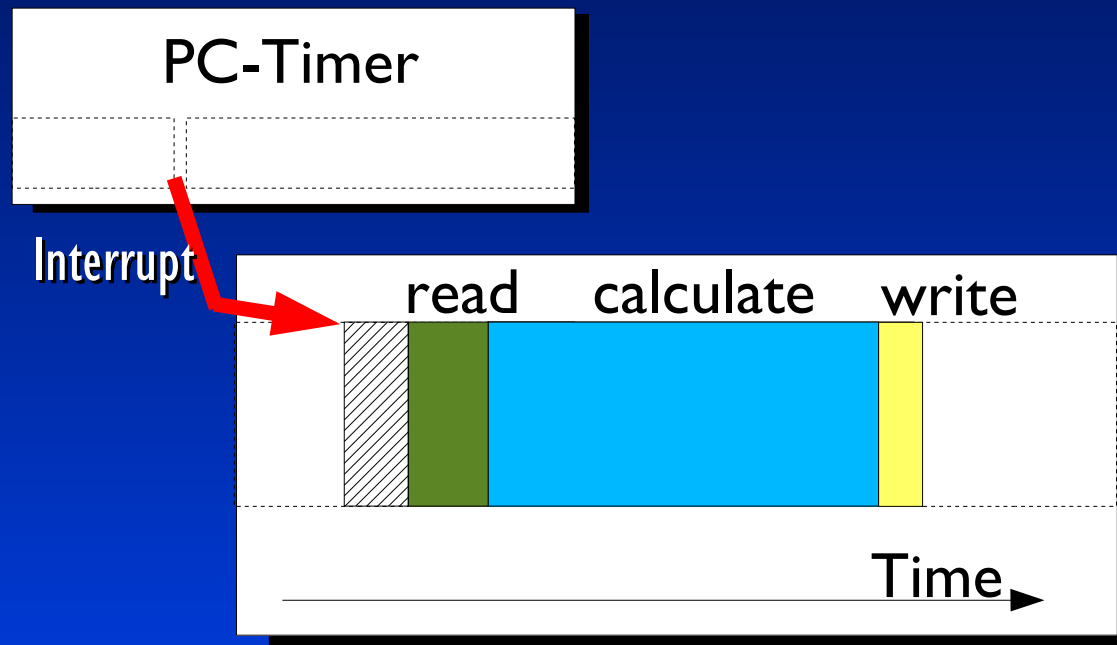
New System Architecture



Aufrufmechanismen I/2

- Jitter

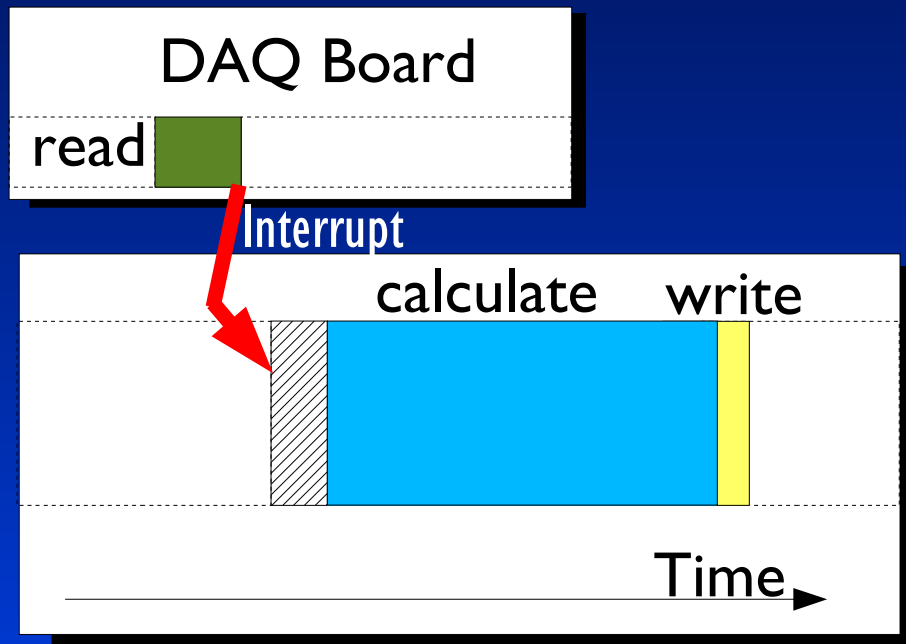
- Bus blockiert, Caches (TLB), CPU Pipeline



- Kompensieren durch Warten (kostet CPU-Zeit)

Aufrufmechanismen 2/2

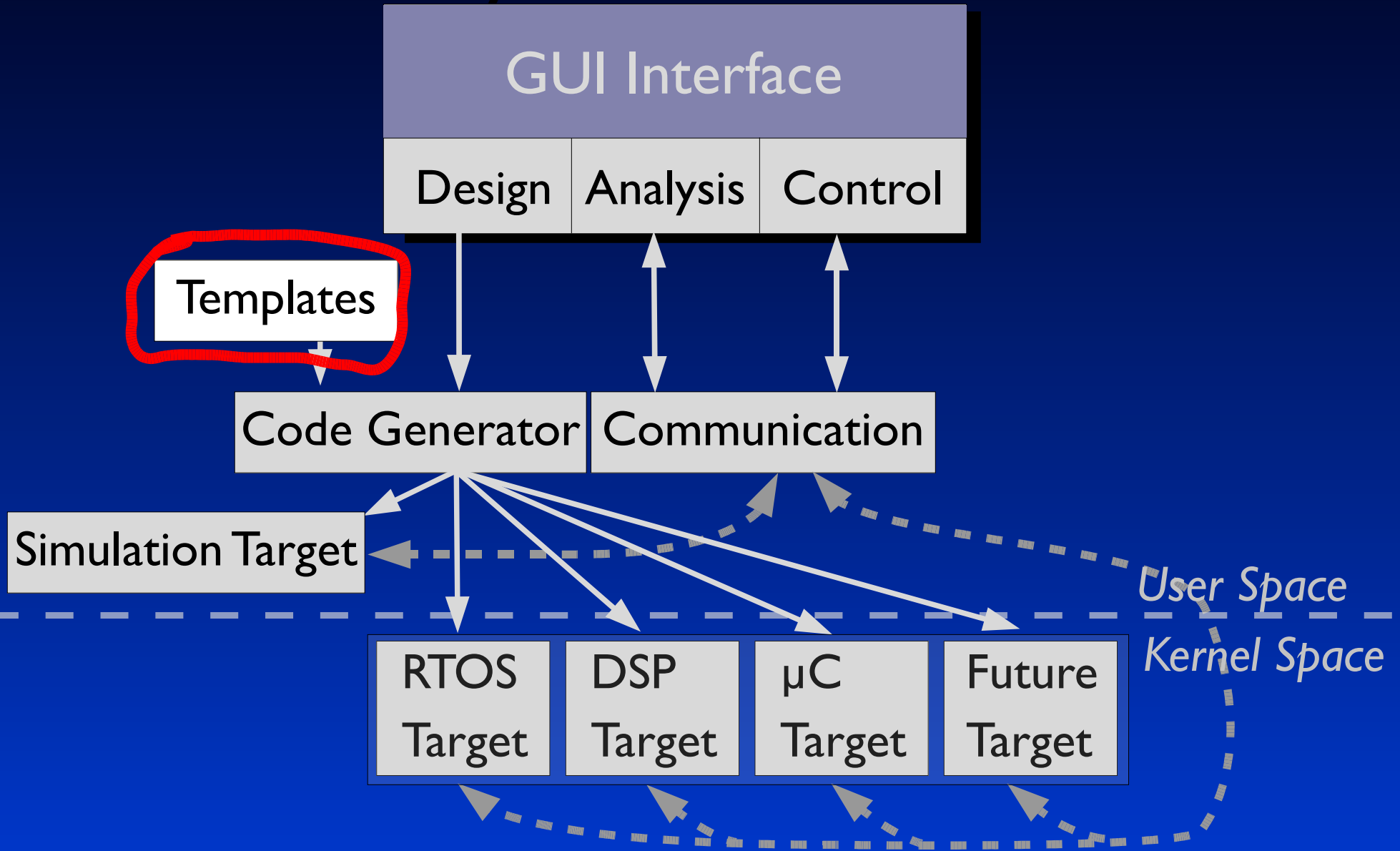
- DAQ Acquisition
 - Zeit für A/D-Wandlung gespart



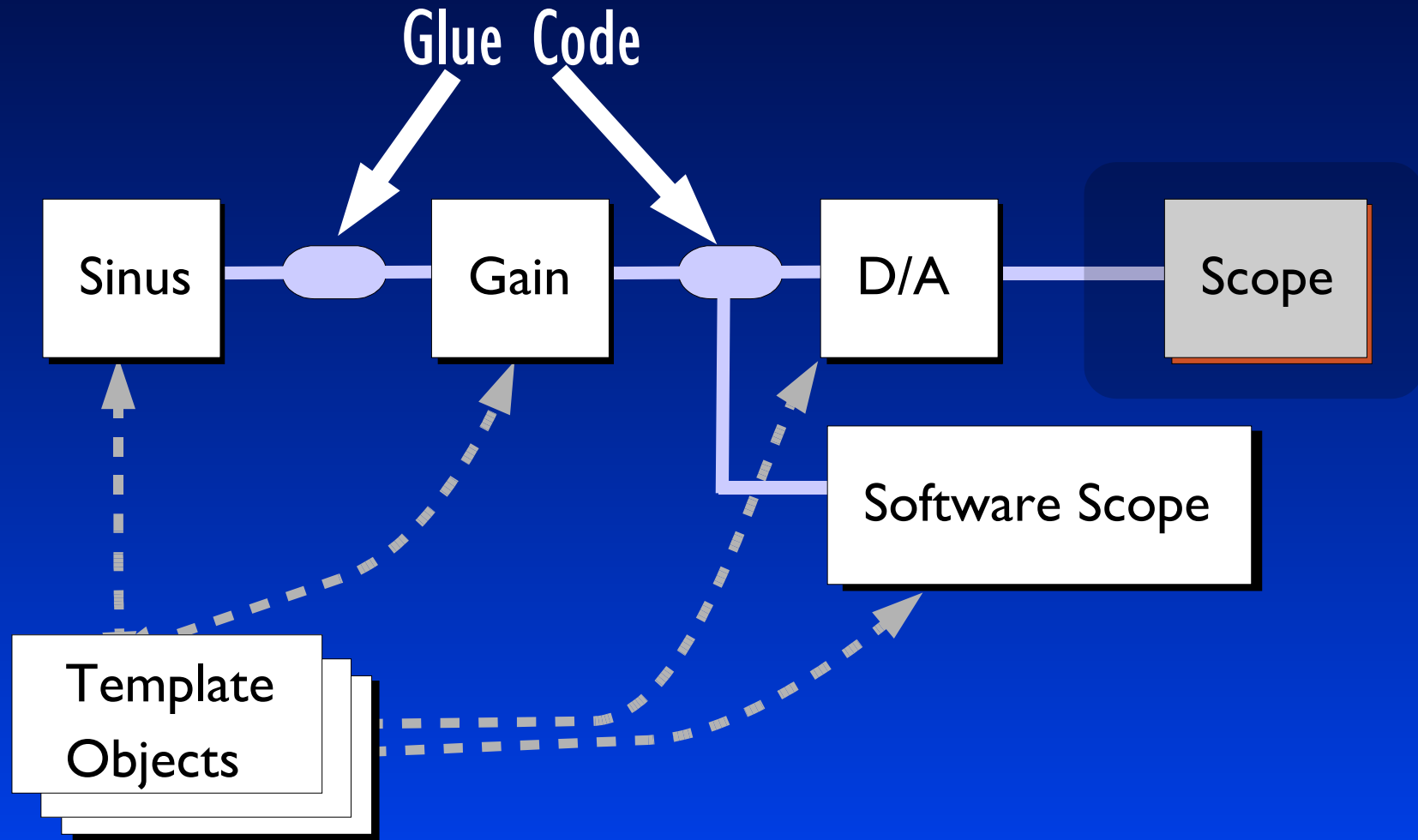
- schreiben synchron
- schreiben asynchron

- Meßwerte: Jitter verringert

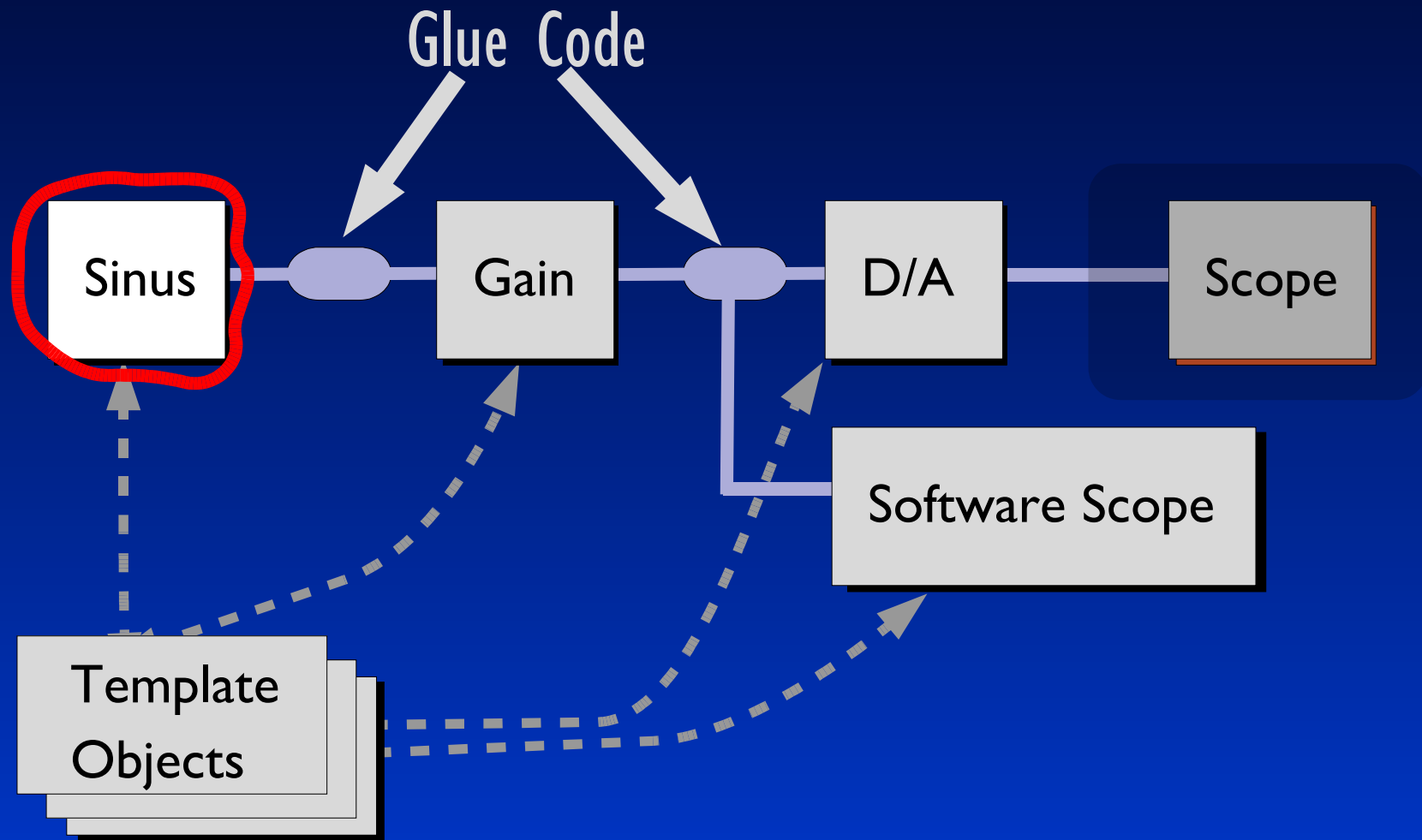
New System Architecture



Codegenerator/Templates



Codegenerator/Templates



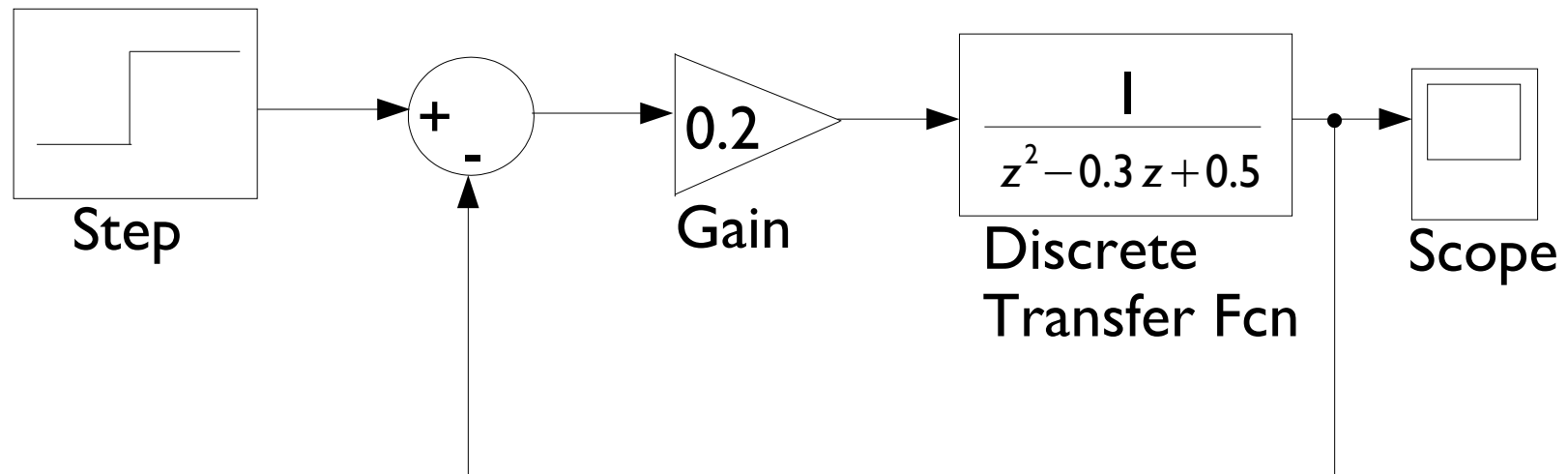
Codegenerator/Sinus-Template 1/2

```
|  
amplitude=1  
omega=1  
phase=0  
step=0.1  
|  
|Section("include")  
#include <math.h>  
  
|Section("update")|  
static double t=0;  
t+=|step|;  
|Output()|=|amplitude|*  
sin(fmod(|omega|*t+  
|phase|,2*PI));
```

Codegenerator/Sinus-Template 2/2

```
(...)  
/* sinus section include */  
#include <math.h> (...)  
  
/* sinus section init */  
sinus1=0.0; (...)  
  
(...)  
{  
/* sinus1 section update */  
static double t=0;  
t+=0.1;  
sinus1=1*sin(fmod(1*t+0,2*PI));  
}  
(...)
```

Beispiel 1/3



Beispiel 2/2

```
in1=t_step.Create(trigger=0)
minus1=t_minus.Create()
plant1=t_plant.Create(a=[1,-0.3,0.5],b=[0,0,1])
gain1=t_gain.Create(gain=0.2)
out1=t_dump.Create()
```

```
tree=TreeNode(in1)
main1=t_main.Create(t=tree,steps=100)
# t_rtlinux.Create(t=tree, khz=1)
```

```
tree.Link([in1],[minus1])
tree.Link([minus1],[gain1])
tree.Link([gain1],[plant1])
tree.Link([plant1],[out1,minus1])
tree.WalkTree()
```

```
source=main1.Gen("main"); PrettyPrint(source);
```

Beispiel 3/3

```
/* plant1 section init */  
plant1_a[0]=1; plant1_a[1]=-0.3;  
(...)
```

```
int main() {
```

```
    init();
```

```
    while(steps--) {
```

```
        { /* in1 section update */
```

```
            static int in1_counter=0; in1_counter++;
```

```
            if (in1_counter>=0) {in1=1;} else {in1=0;}
```

```
        }
```

```
        { /* minus1 section update */
```

```
            minus1=in1-plant1[0];
```

```
        }
```

```
        { /* plant1 section update */
```

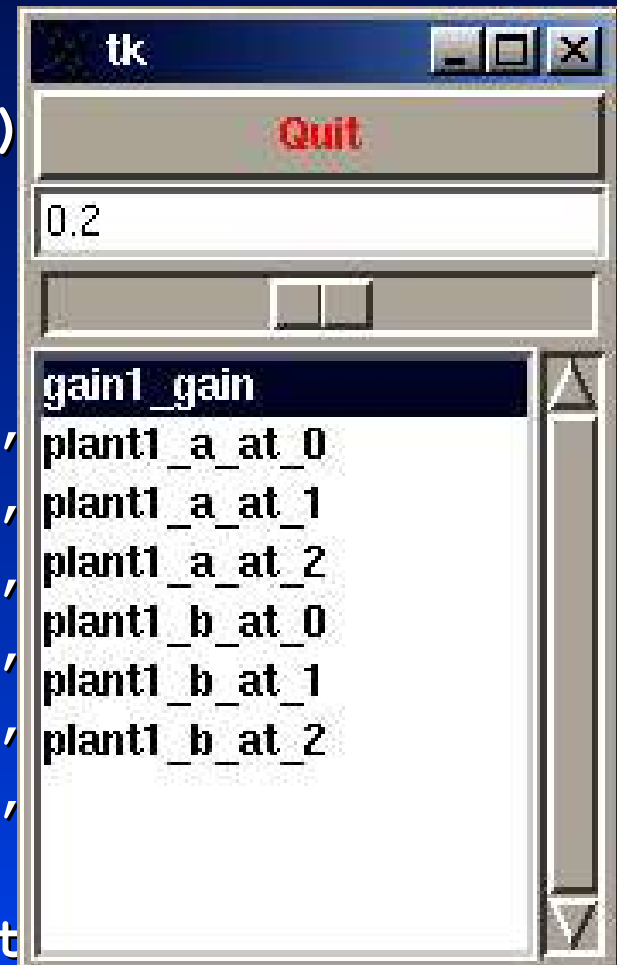
```
            memmove(plant1+1,plant1,
```

```
                sizeof(double)*(2));
```

```
            (...)
```

Codegenerator/Coating

```
struct arg_register_t
arg_register[]={
  {magic, ((double*)&arg_register)
  /* gain1 section register */
  {"gain1_gain", &gain1_gain},
  /* plant1 section register */
  {"plant1_a_at_0", &plant1_a[0]},
  {"plant1_a_at_1", &plant1_a[1]},
  {"plant1_a_at_2", &plant1_a[2]},
  {"plant1_b_at_0", &plant1_a[0]},
  {"plant1_b_at_1", &plant1_a[1]},
  {"plant1_b_at_2", &plant1_a[2]},
  {"", NULL}};
EXPORT_SYMBOL_NOVERS(arg_register
```



Codegenerator/Coating

```
struct arg_register t
```

```
ar
```

```
{m
```

```
/*
```

```
{ "
```

```
/*
```

```
{ "
```

```
{ "
```

```
{ "
```

```
{ "
```

```
{ "
```

```
{ "
```

```
{ "plant1_b_at_2", &plant1_a[2] },
```

```
{ "", NULL } };
```

```
EXPORT_SYMBOL_NOVERS (arg_regist
```

Normal

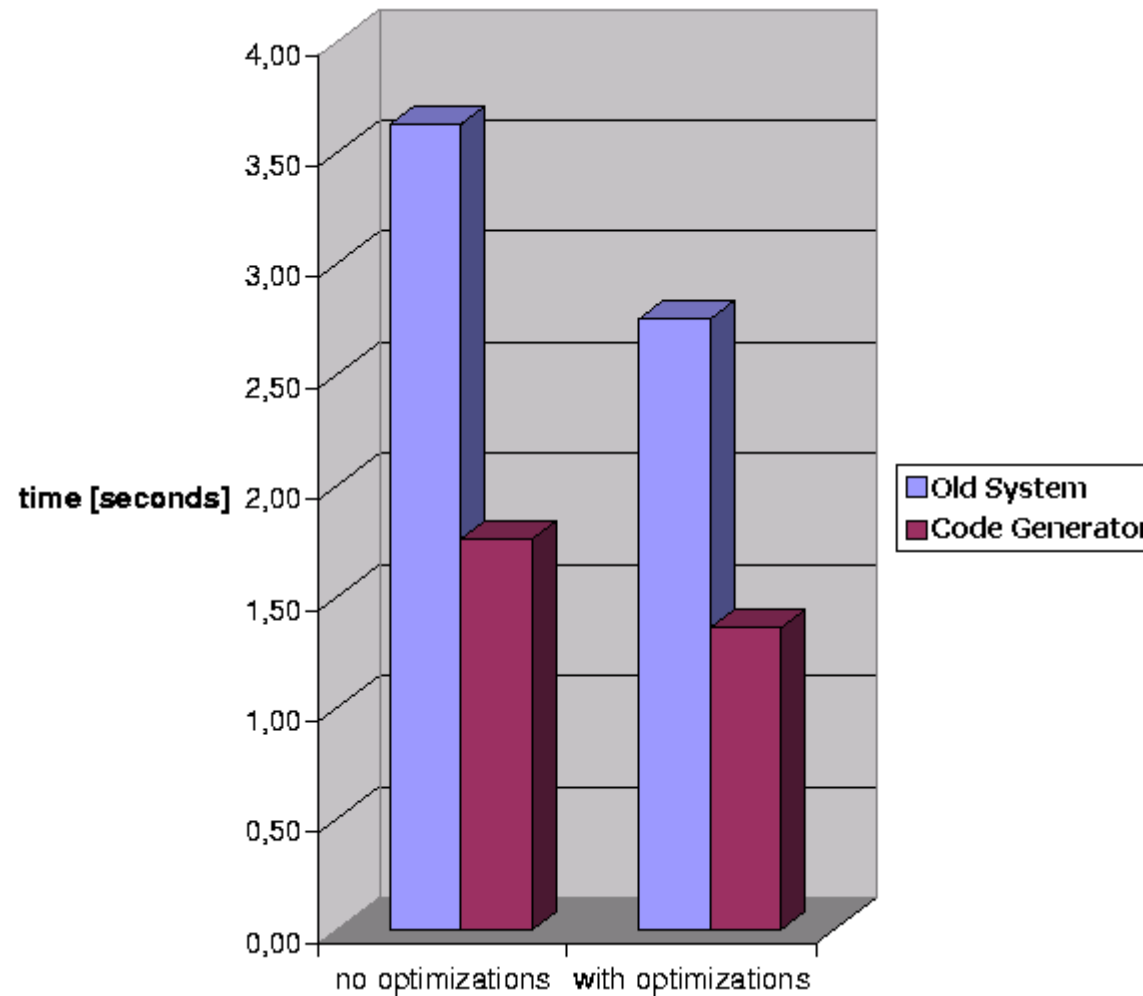
- alle Parameter fix im Quellcode

mit "Coating"

- bestimmte Parameter variabel
- Gesamtlaufzeitkosten trotzdem gering



Geschwindigkeitsvergleich



Geschwindigkeitsvergleich

Fazit:

- Asynchroner Aufrufmechanismus $\approx 2x$
- Codegenerator $\approx 2x$
- absturzresistenter
- geeigneter für eingebettete Systeme
- sprachunabhängig (C++, C, Assembler)
- automatische Konfiguration

0,00

no optimizations | with optimizations

Aussicht

Codegenerator

- Vererbung und Templates (z.B. Spezialisierung)
- Typ- und Schnittstellenprüfung
- Automatische Konfiguration
(User- vs. Kernel-Space, Hardware)
- “On the fly” Rekonfiguration und
Recompilierung