

CiAO (CiAO is Aspect-Oriented)

Eine aspektorientiert entworfene Betriebssystemfamilie

Daniel Lohmann

Olaf Spinczyk

Wolfgang Schröder-Preikschat

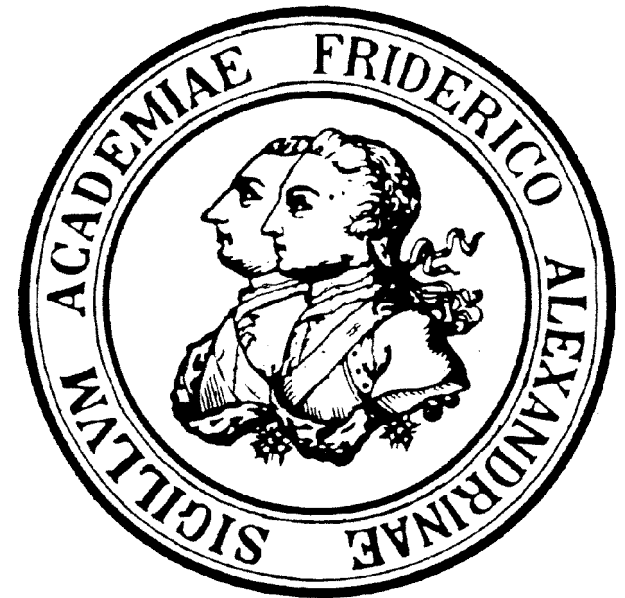
Lehrstuhl für Informatik IV

Verteilte Systeme und Betriebssysteme

Friedrich-Alexander Universität Erlangen-Nürnberg

lohmann@cs.fau.de

<http://www4.cs.fau.de/~lohmann>



Agenda

- Das CiAO-Projekt: Überblick
- Problemfeld und Hintergrund
- Ansatz
- Beispiel
- Zusammenfassung



Das CiAO – Projekt: Überblick

- DFG-gefördertes BS-Forschungsprojekt (SCHR 603/4)
 - Eine WM-Stelle, zwei Studenten, seit Oktober 2004
- Projektziele
 - Entwicklung einer hochgradig anwendungsgewahren Betriebssystem-Produktlinie für tief eingebettete Systeme
 - Gute **Anpassbarkeit** an die Anforderungen der Anwendung
 - Geringer **Ressourcenbedarf**
 - Entwicklung eines **wandlungsfähigen** Betriebssystems
 - Komponentenimplementierung (Treiber, Scheduler, ...)
unabhängig von der Architektur (monolithisch, Mikrokern, Bibliothek, ...)
 - **Konfigurierbare** nicht-funktionale Architektureigenschaften
 - ... durch gezielte **Verwendung von AOP**



Das CiAO – Projekt: Überblick

- Wissenschaftliche Fragestellung
 - Sind Aspekttechniken **zielführend** bei der Produktlinienentwicklung?
 - Lässt sich durch AOP ein (deutlich) **höheres Maß** an Konfigurierbarkeit erreichen?
 - Lassen sich damit insbesondere auch **nicht-funktionale Eigenschaften / Architektureigenschaften** konfigurierbar gestalten?
 - Wie sehen **architekturtransparente** BS-Komponenten aus?



Problemfeld: Anwendungsanpassbarkeit

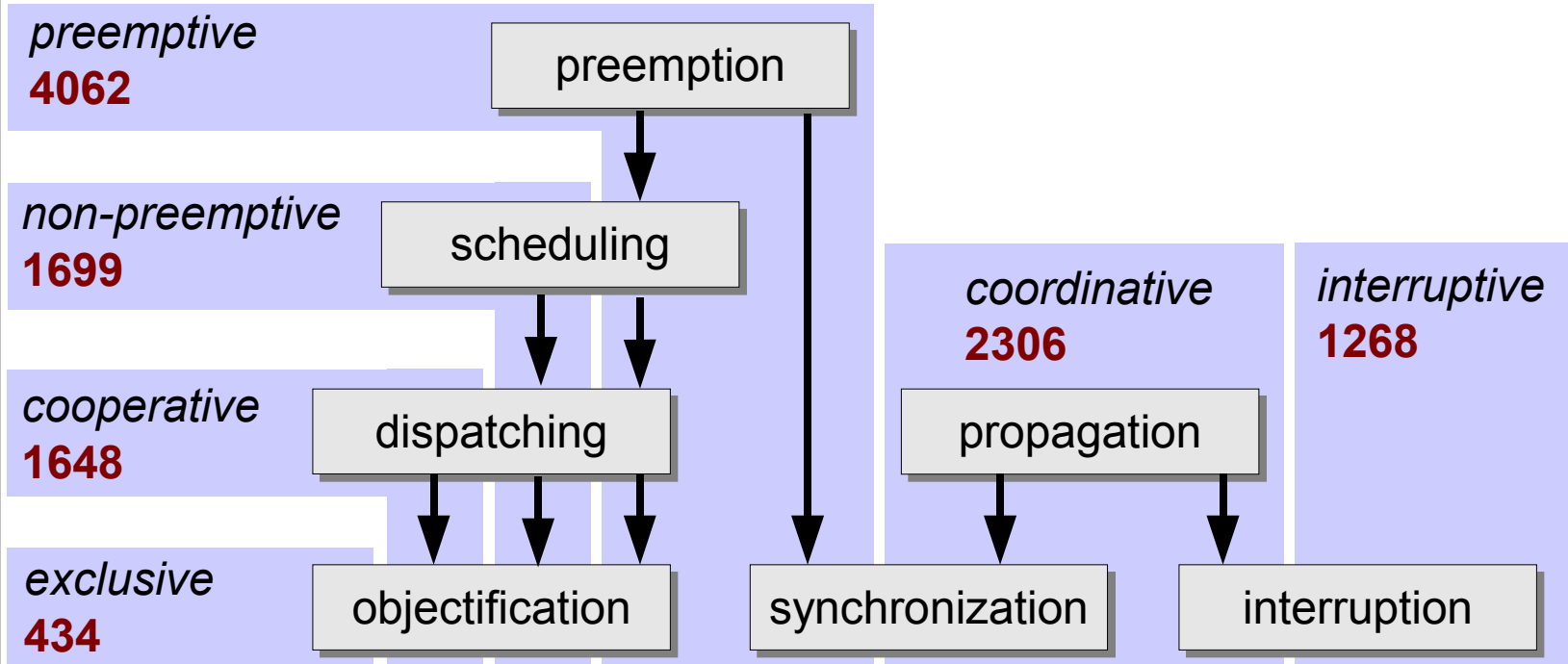
Zwei Dimensionen der Anwendungsanpassbarkeit

- Granularität (Funktionale Auswählbarkeit)
 - „Ich muss nichts einbinden, was ich nicht brauche“
 - Resultierende Systeme müssen klein sein!
 - Geringstmögliche Anforderungen an die Hardware
- Variabilität (Funktionale Anpassbarkeit)
 - Verschiedenste Plattformen (8-64 Bit, mit/ohne MMU)
 - Alle Eigenschaften lassen sich konfigurieren / anpassen



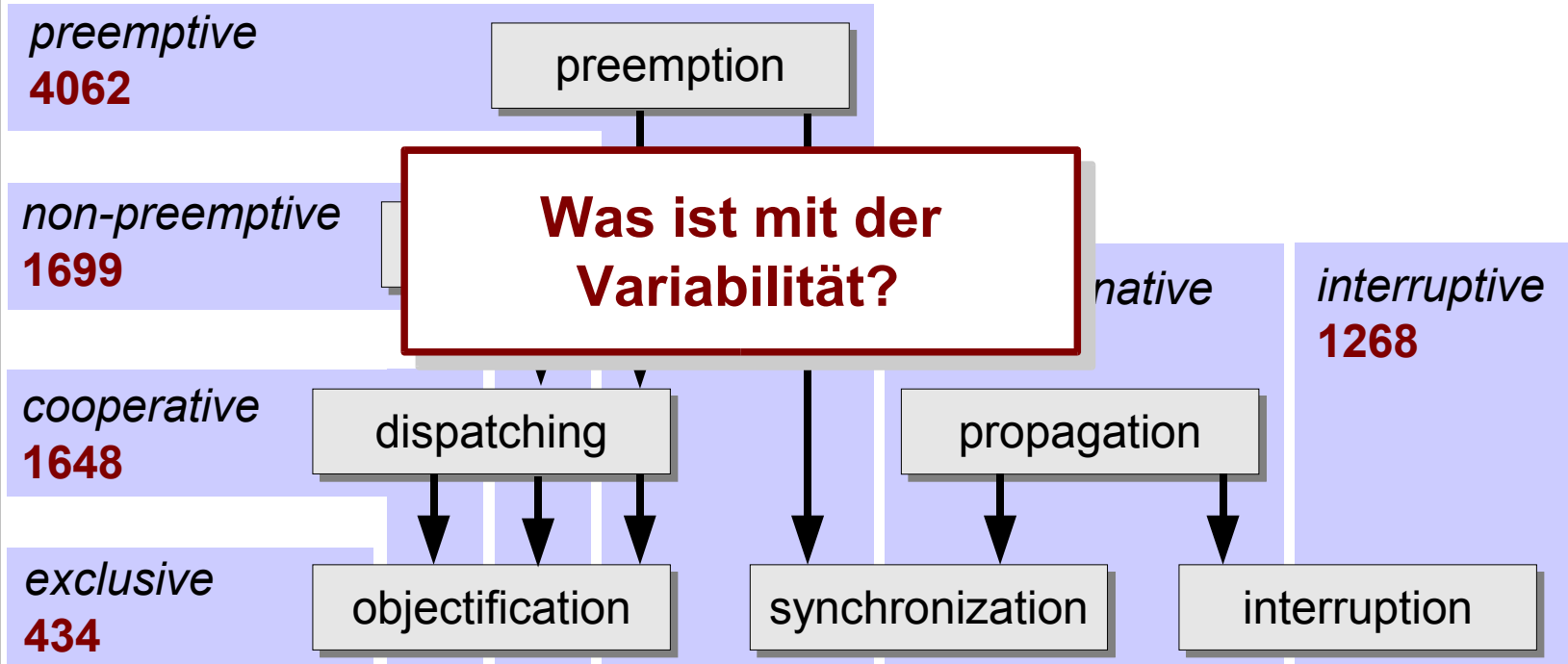
Historie: Der PURE Nucleus

- 1995-2003, Uni Potsdam/Magdeburg
- feinere Granularität geht wohl kaum ;-)

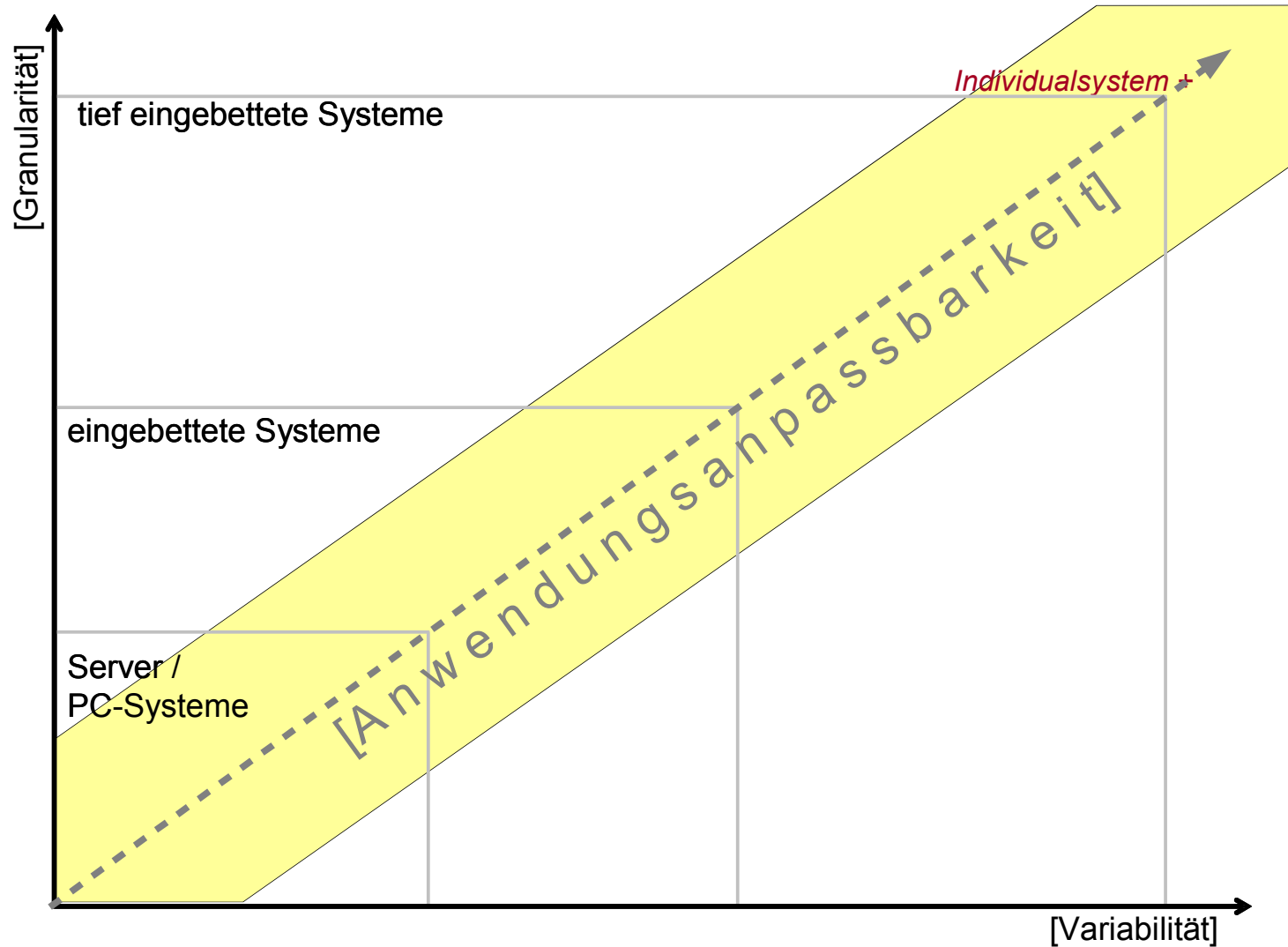


Historie: Der PURE Nucleus

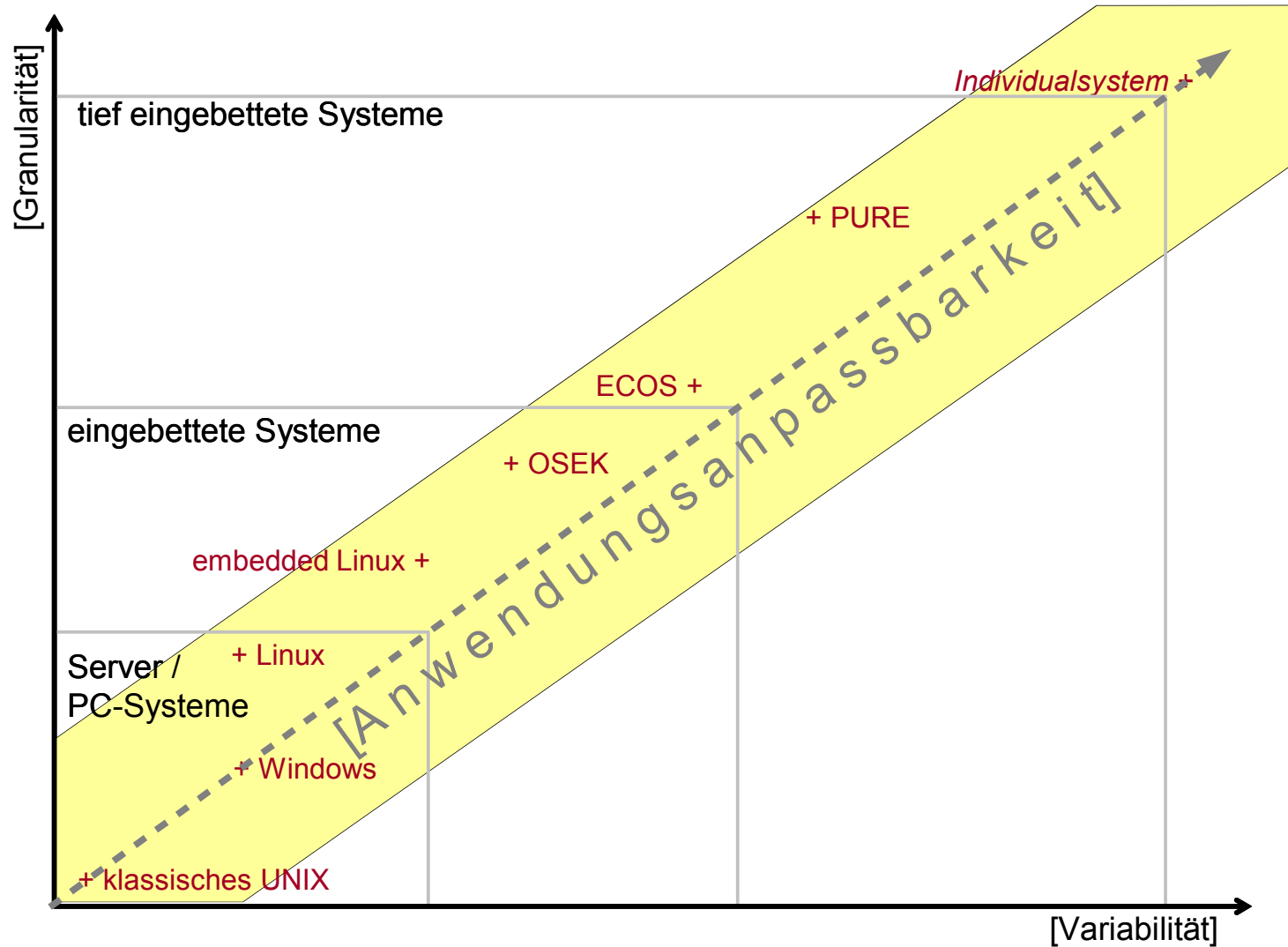
- 1995-2003, Uni Potsdam/Magdeburg
- feinere Granularität geht wohl kaum ;-)



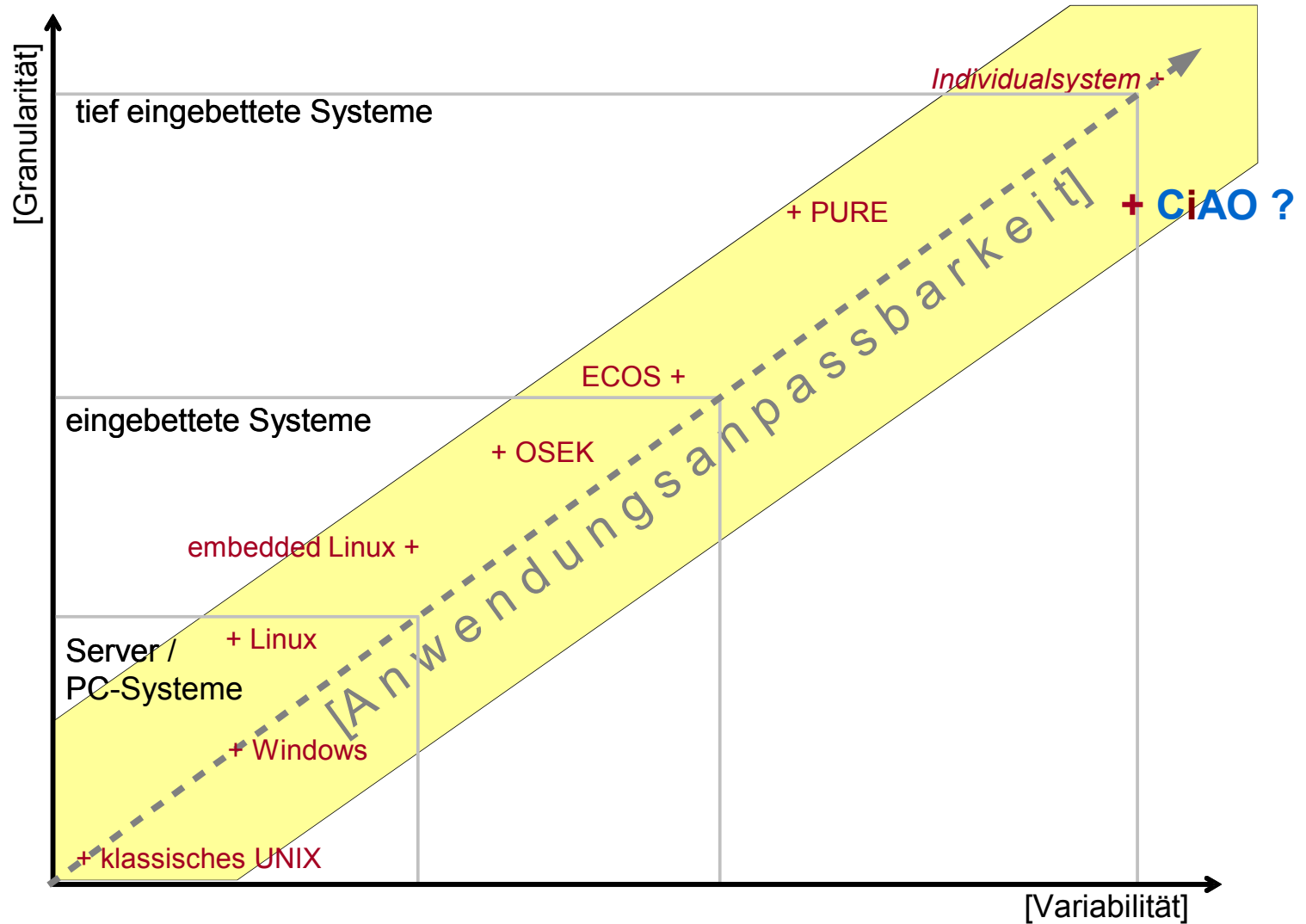
Granularität und Variabilität



Granularität und Variabilität



Granularität und Variabilität



Im Fokus von CiAO: Variabilität

- Konfigurierung **nicht-funktionaler Eigenschaften**
 - Kernsynchronisation
 - grobgranular, feingranular, ohne
 - Unterbrechungsbehandlung
 - Verteilung (Hardware, Scheduler)
 - Bearbeitung (Handlerfunktion, Prolog-/Epilog, Kernelthread, Prozess)
 - Speicherschutzkonzept
 - ohne, Segmentierung, Addressräume
 - Platzierung der BS-Komponenten
 - gemeinsam, Segmente, Addressräume, Prozesse
 - Interaktion zwischen BS-Komponenten
 - Prozeduraufruf, Nachricht, IPC
 - Optimierungen
 - Caching, Aufrufpfade



Im Fokus von CiAO: Variabilität

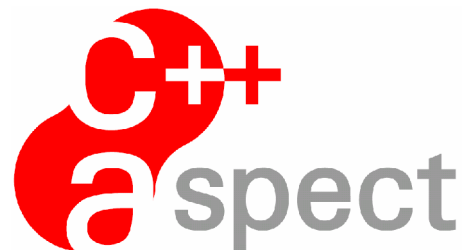
- Konfigurierung **nicht-funktionaler Eigenschaften**
 - Kernsynchronisation
 - grobgranular, feingranular, ohne
 - Unterbrechungsbehandlung
 - Verteilung (Hardware, Scheduler)
 - Bearbeitung (Handlerfunktion, Prolog-/Epilog, Kernelthread, Prozess)
 - Speicherschutzkonzept
 - ohne, Seg
 - Platzierung
 - gemeinsam
 - Interaktion zwischen BS-Komponenten
 - Prozeduraufruf, Nachricht, IPC
 - Optimierungen
 - Caching, Aufrufpfade

Konfigurierbare Architektur



Ansatz: Verwendung von AOP

- zur Implementierung **querschneidender Belange**
 - Strategien (Synchronisation, Ressourcenmanagement...)
 - Plattformspezifische Optimierungen (Caching, Bypassing)
- zur **Bindung/Interaktion** der Komponenten
 - Je nach Zielarchitektur (Prozedur, Moduswechsel, IPC)
 - Variabilität und Granularität durch lose Kopplung
- zur **Platzierung** der Komponenten
 - Je nach Zielarchitektur (Addressraum, Thread, Prozess)



Ansatz: Verwendung von AOP

- zur Implementierung **querschneidender Belange**
 - Strategien (Synchronisation, Ressourcenmanagement...)
 - Plattformspezifische Optimierungen (Caching, Bypassing)
- zur **Bindung/Interaktion** der Komponenten

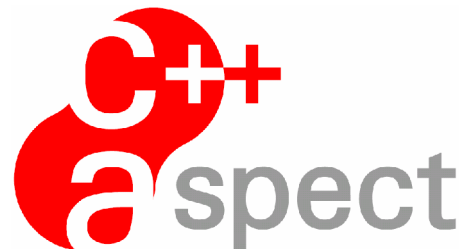
- Je

- V

Komponenten müssen aspektgewahr entworfen sein

- zu

- Je nach Zielarchitektur (Addressraum, Thread, Prozess)



RS232-Treiber, direkter Zugriff

```
class RS232 {  
    ...  
public:  
    void read(char*, int) {  
        ...  
        ...  
    }  
    ...  
};
```

```
int main() {  
    ...  
    RS232 com;  
    com.read(buf, 16);  
    ...  
}
```

RS232-Treiber, konfiguriert als *ProtectedComponent*

```
class RS232 {
    ...
public:
    void read(char*, int) {
        ...
    }
    ...
};
```

```
int main() {
    ...
    RS232 com;
    com.read(buf, 16);
    ...
}
```

```
class Mutex {
    ...
    void lock();
    void unlock();
};

aspect ProtectedComponent {
    pointcut virtual classes() = 0;
    pointcut virtual funcs() = 0;

    advice classes() : Mutex _m;

    advice execution(funcs())
    : before {
        tjp->that()->_m.lock();
    }

    advice execution(funcs())
    : after {
        tjp->that()->_m.unlock();
    }
};
```

RS232-Treiber, konfiguriert als *ProtectedComponent*

```
class RS232 {
    ...
public:
    void read(char*, int) {
        ...
        ...
    }
    ...
};
```

```
aspect RS232Config
: public ProtectedComponent {

    pointcut classes() = „RS232“;
    pointcut funcs() = „,% RS232::%(...)“;
};
```

```
int main() {
    ...
    RS232 com;
    com.read(buf, 16);
    ...
}
```

```
class Mutex {
    ...
    void lock();
    void unlock()
};

aspect ProtectedComponent {
    pointcut virtual classes() = 0;
    pointcut virtual funcs() = 0;

    advice classes() : Mutex _m;

    advice execution(funcs())
    : before {
        tjp->that()->_m.lock();
    }

    advice execution(funcs())
    : after {
        tjp->that()->_m.unlock();
    }
};
```

RS232-Treiber, konfiguriert als *ProtectedComponent*

```
class RS232 {  
    Mutex _m;  
public:  
    void read(char*, int) {  
        __before_1();  
        ...  
        __after_1();  
    }  
    ...  
};
```

```
aspect RS232Config  
: public ProtectedComponent {  
  
    pointcut classes() = „RS232“;  
    pointcut funcs() = „,% RS232::%(...)“;  
};
```

```
int main() {  
    ...  
    RS232 com;  
    com.read(buf, 16);  
    ...  
}
```

```
class Mutex {  
    ...  
    void lock();  
    void unlock()  
};  
  
aspect ProtectedComponent {  
    pointcut virtual classes() = 0;  
    pointcut virtual funcs() = 0;  
  
    advice classes() : Mutex _m;  
  
    advice execution(funcs())  
    : before {  
        tjp->that()->_m.lock();  
    }  
  
    advice execution(funcs())  
    : after {  
        tjp->that()->_m.unlock();  
    }  
};
```

RS232-Treiber, konfiguriert als *MiniServer*

```
class RS232 {
    ...
public:
    void read(char*, int) {
        ...
        ...
    }
    ...
};
```

```
int main() {
    ...
    RS232 com;
    com.read(buf, 16);
    ...
}
```

```
class MiniServer : public Thread {
    ...
    void run() {
        AC::Action* act;
        while(true){
            Thread& s=receive(ANY, act);
            act->trigger();
            send(s);
        }
    }
};

aspect MiniServerComponent {
    pointcut virtual classes() = 0;
    pointcut virtual funcs() = 0;

    advice classes() : MiniServer _s;

    advice call(funcs())
    && !within(classes()) : around(){
        send(tjp->target()->_s,
            &tjp->action());
        receive(tjp->target()->_s);
    }
};
```

RS232-Treiber, konfiguriert als *MiniServer*

```
class RS232 {  
    MiniServer _s;  
public:  
    void read(char*, int) {  
        ...  
        ...  
    }  
    ...  
};
```

```
aspect RS232Config  
: public MiniServerComponent {  
  
    pointcut classes() = „RS232“;  
    pointcut funcs() = „,% RS232::%(...)“;  
};
```

```
int main() {  
    ...  
    RS232 com;  
    __around_1( <com.read(buf, 16)> );  
    ...  
}
```

```
class MiniServer : public Thread {  
    ...  
    void run() {  
        AC::Action* act;  
        while(true){  
            Thread& s=receive(ANY, act);  
            act->trigger();  
            send(s);  
        }  
    }  
};  
  
aspect MiniServerComponent {  
    pointcut virtual classes() = 0;  
    pointcut virtual funcs() = 0;  
  
    advice classes() : MiniServer _s;  
  
    advice call(funcs())  
    && !within(classes()) : around(){  
        send(tjp->target()->_s,  
            &tjp->action());  
        receive(tjp->target()->_s);  
    }  
};
```

Zusammenfassung: Das CiAO-Projekt

- Evaluierung von AOP im Betriebssystembau
 - AspectC++
 - Konfigurierung von Betriebssystem-Produktlinien
 - Insbesondere nicht-funktionale Eigenschaften
- Wandlungsfähiges Betriebssystem
 - Architekturtransparente Komponenten
 - Durch Aspekte an konkrete Architektur angepasst
- Ziel: Hohe Variabilität
 - Grenzen der Konfigurierbarkeit
 - Kosten (Größe/Laufzeitoverhead) der Konfigurierbarkeit

