SPIN Operating System

- Motivation und Ziele von SPIN
- Aufbau und Architektur
- Erweiterbarkeit durch Event-Mechanismus
- Sicherheit
- Zusammenfassung

Motivation

- **radikaler Wandel auf der Anwendungsebene**
  früher: Workstations für wissenschaftliche Berechnungen
  heute: Multi-Media, verteilte Datenbanken

- **Zumeist veraltete Konzepte in heutigen Betriebssystemen**
  Anwendung muß sich auf das Betriebssystem einstellen

- **Gratwanderung zwischen Generalisierung und Spezialisierung**
  Ziel: Anwendung sollen das Betriebssystem an eigene Bedürfnisse anpassen können

Ziele von SPIN

- **Erweiterbarkeit**

  Anwendungen sollen System dynamisch um spezialisierte Dienste erweitern können

- **Sicherheit**

  Kernel soll von Aktionen der Erweiterungen geschützt werden

- **Performance**

  Kosten von Erweiterbarkeit und Sicherheit möglichst gering halten

Der Ansatz von SPIN

- **Co-location**

  Dynamisches Linken von Betriebssystemerweiterungen in den Adreßraum des Kerns

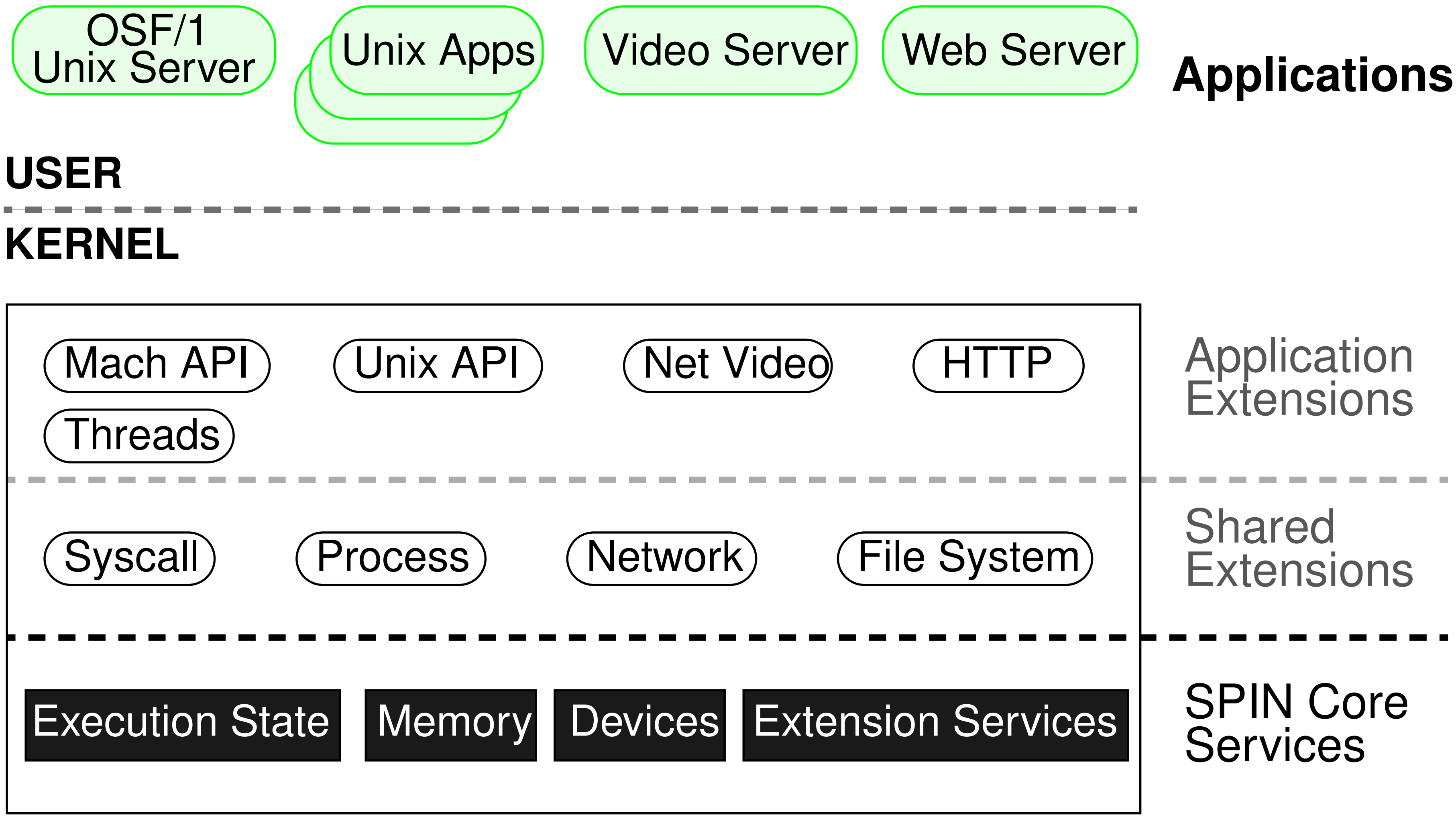
- **Erzwungene Modularität**

  Erweiterungen müssen in Modula-3 geschrieben werden

- **Logical Protection Domains**
  - Namensräume innerhalb des Kerns
  - umschließen eine Menge von Interfaces

- **Dynamic Call Binding**
  - Erweiterungen laufen immer als Antwort auf System-Events ab
  - Event: Jede Zustandsänderung im System

Der Aufbau von SPIN



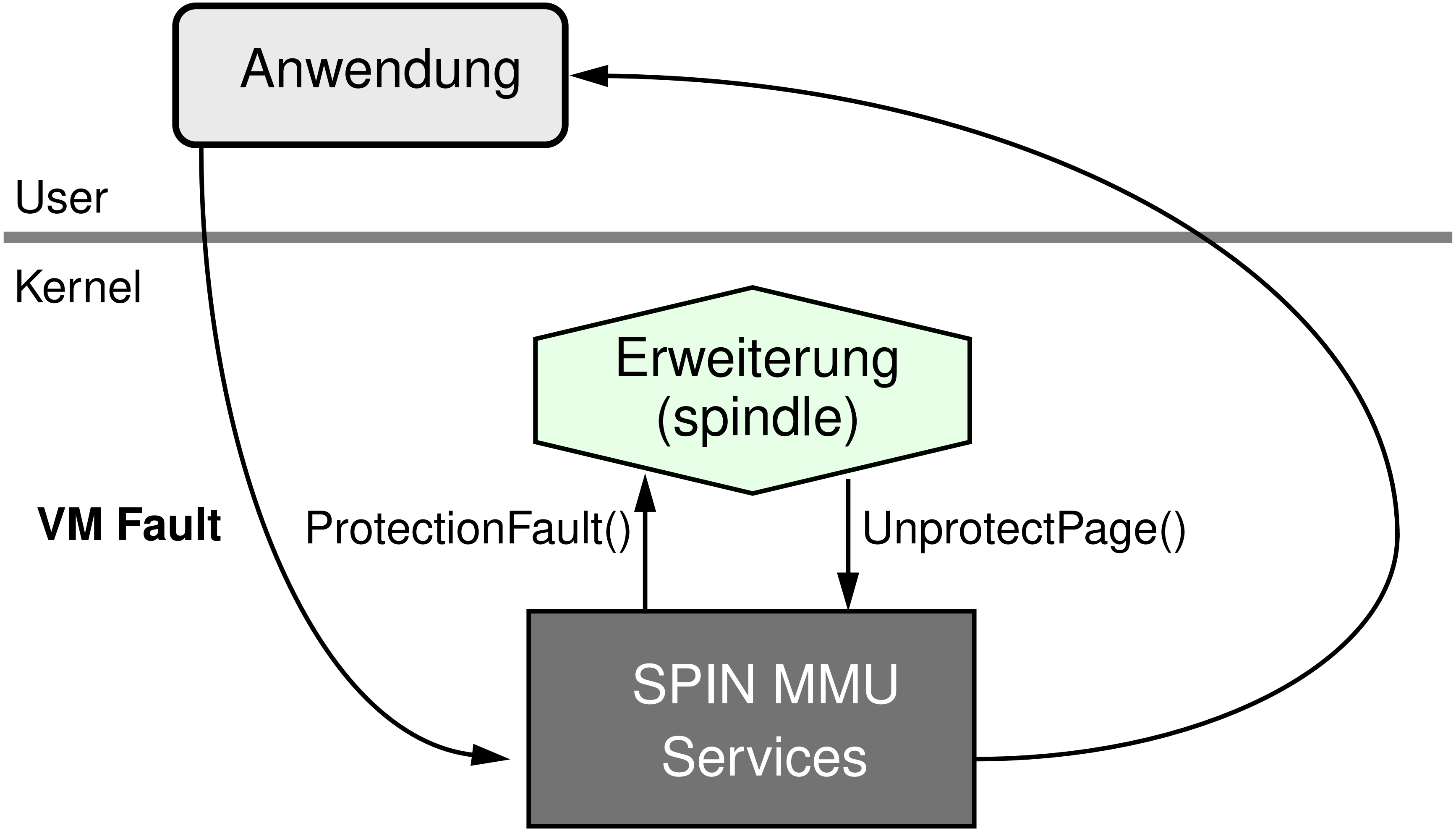
Der Mikrokernel

- **Mikrokernel bietet nur grundlegende Dienste an:**
  - Thread-Management
  - Device-Management
  - Virtual-Memory-Management
  - Erweiterungsdienste

- **Strikte Trennung zwischen Mechanismus und Policy**

- **SPINDLES (SPIN Dynamically Loaded Extensions) implementieren anwendungsspezifische Dienste / Policies**
  - Beispiele: FileSystem, User-Level-Threads, Netzwerk-Dienste

SPIN-Erweiterung (SPINDLE)

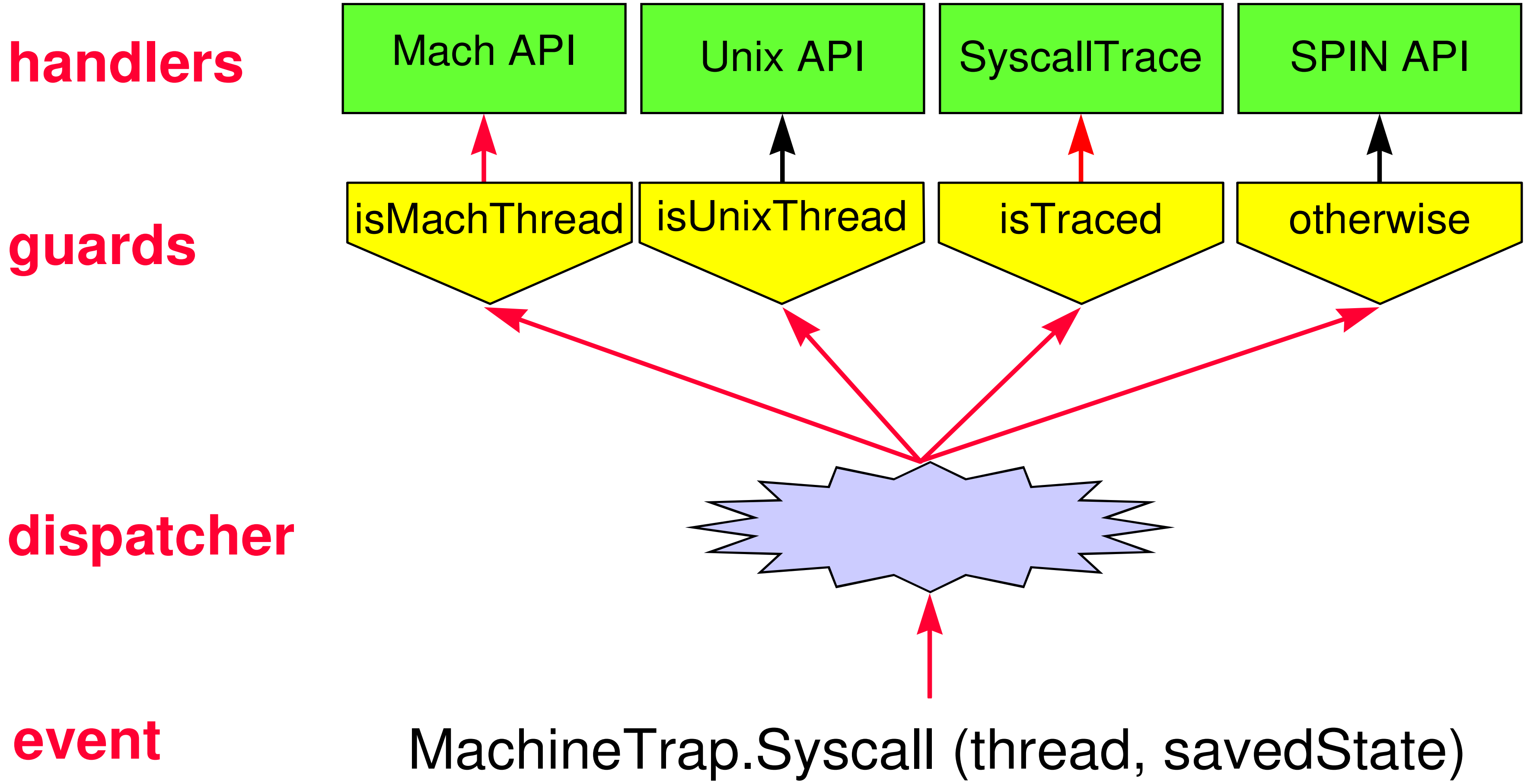


Erweiterbarkeit durch SPINs Event-Mechanismus

Events = Prozeduren + Flexibilität

- ❏ **Jede Modula-3 Prozedur des Kerns oder einer Erweiterung stellt ein potentielles Event dar**
  - einfache Syntax, hohe Performance
- ❏ **Erweiterte Prozedursemantik**
  - Multi-Cast, bedingte Ausführung, Asynchronität
- ❏ **Event-Raising = Aufrufen einer Prozedur**
- ❏ **Event-Handling = Ausführen einer Prozedur**
- ❏ **Event-Dispatcher: Verteilung von Events vom Event-Raiser zu den Event-Handlern**

The life of a system call



Programmer's View

Event definition

```
INTERFACE MachineTrap;
PROCEDURE Syscall(
            thread  : Strand.T;
            state   : SavedState);
END MachineTrap.
```

Event invocation

```
MODULE MachineTrapPrivate;
PROCEDURE Syscall(thread, state) =
  BEGIN
    …
    MachineTrap.Syscall(thread,state)
    …
  END Syscall;
END MachineTrapPrivate.
```

Extension

```
MODULE MachEmulation;
PROCEDURE Handler (thread, state) =
  CASE state.v0 OF
       …
       | -65 => VM.vm_allocate();
       …
  END Handler;
PROCEDURE Guard (thread, state)
                        : BOOLEAN =
       RETURN IsMachThread(thread);
  END Guard;
BEGIN
  Dispatcher.InstallHandler(
          MachineTrap.Syscall,
          Guard, Handler);
END MachEmulation.
```

Sicherheit

Die Verwendung von Modula-3 für Erweiterungen und fast alle Kernel-Dienste bietet:

- **Speicherschutz**
  - statische Typprüfung
  - Feldgrenzenüberprüfung (statisch, dynamisch)
- **Interfaces**
  - deklarieren den nach außen sichtbaren Teil eines Moduls
  - Verbergen von Ressourcen
- **Capabilities:**
  - einfache Implementierung durch Zeiger

Dynamisches Linken

Co-location: Der Erweiterungscode einer Anwendung wird dynamisch in den Adreßraum des Kerns gelinkt.

Problem: Wie kann die Typsicherheit der Erweiterung gewährleistet werden ?

Dynamischer Linker akzeptiert eine Erweiterung nur dann, wenn

- der Objektcode der Erweiterung vom Modula-3 Compiler signiert wurde

oder

- die Sicherheit andersweitig garantiert werden kann (z.B. Code ist direkt vom Hersteller)

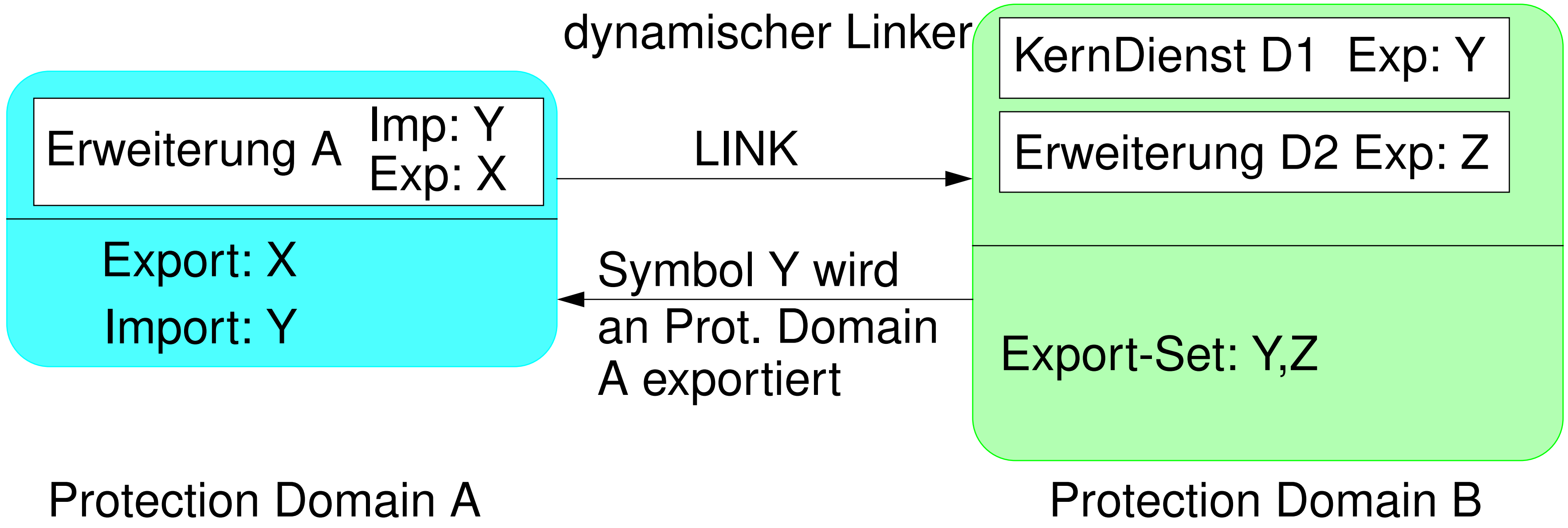
Protection Domains

Ziel: Einfache Zugriffskontrolle für Interfaces

Strategie: - Zugriff nur während des dynamischen Linkens möglich

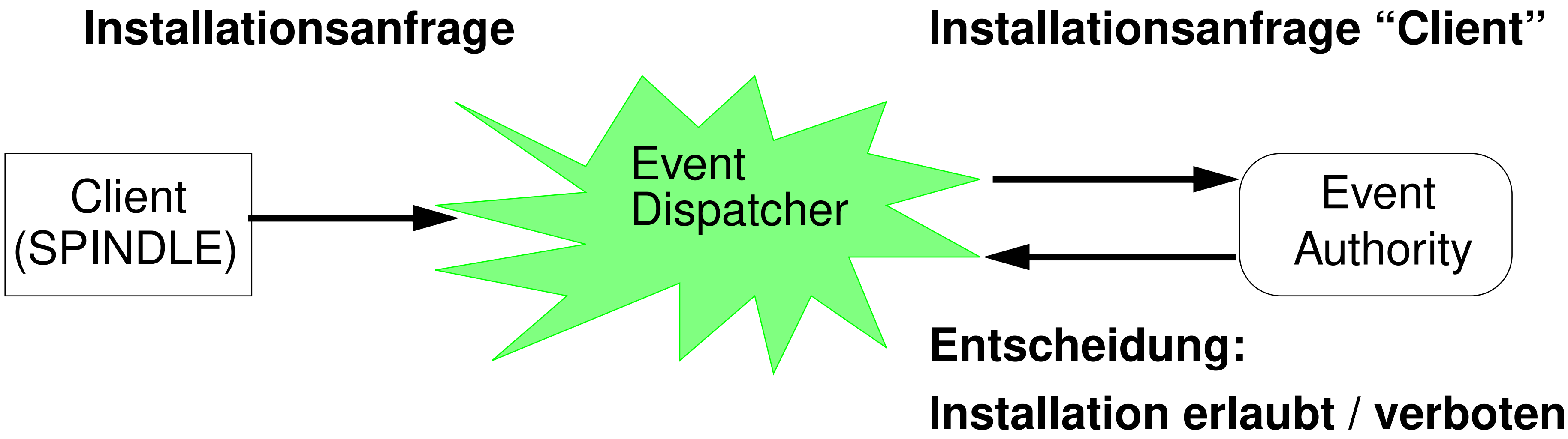
- Zugriffskontrolle durch Protection Domains

Protection Domain: Menge von Interfaces (Symbole, Code)



Installation eines neuen Handlers

- Jedes Event besitzt eine Event-Authority
- Event-Authority erlaubt / verbietet Installation eines Handlers für “sein” Event.

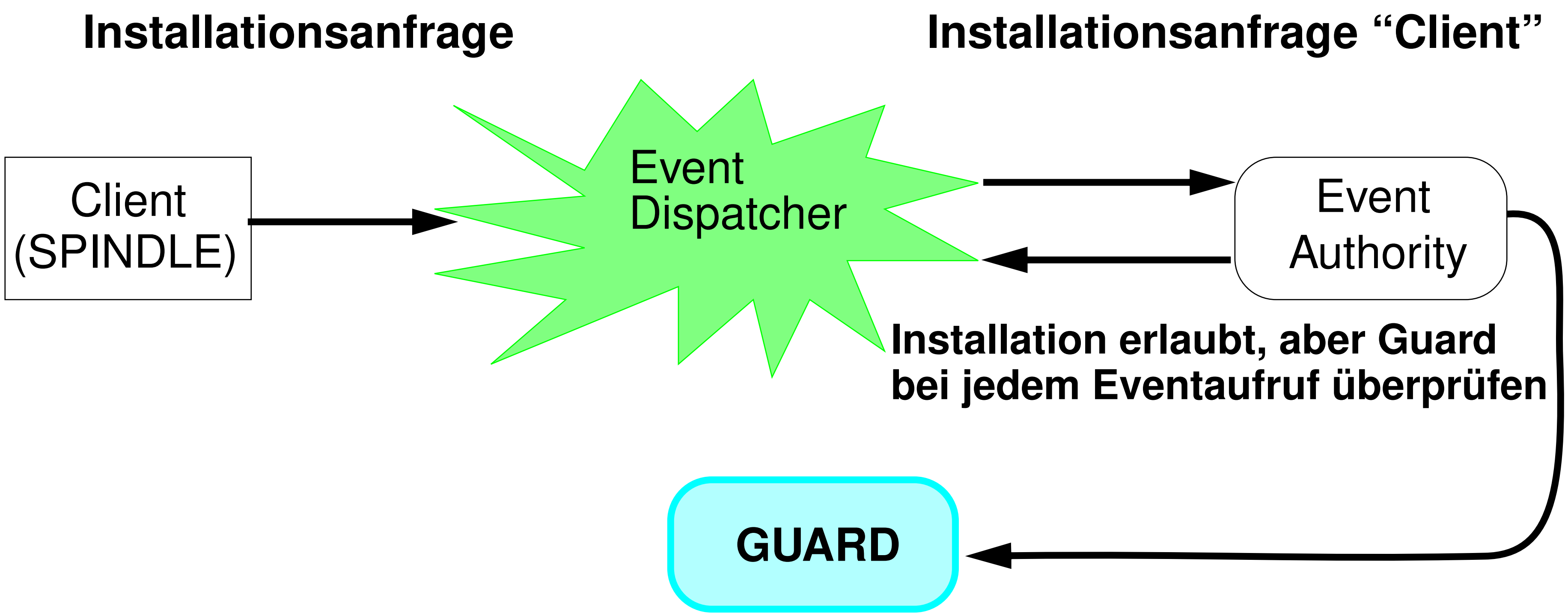


Verteilung von Events an Event-Handler

Problem: Nicht jedes Event darf an jeden Handler weitergeleitet werden

Lösung: Installation eines Guards

➩ dynamische Überprüfung, ob ein Handler ein bestimmtes Event erhalten darf



Run-Away-Handler

Problem: - Ein Handler benötigt zu lange für seine Ausführung

oder

- Ein Handler kommt überhaupt nicht zurück (“läuft davon”)

❑ **Lösung 1 (Präventivmaßnahme):**

Der Handler wird asynchron aufgerufen

➪ teuer, da neuer Thread erzeugt werden muß

❑ **Lösung 2:**

Der Handler wird nach zu langer Laufzeit terminiert

➪ Handler muß Terminierung explizit erlauben

➪ Handler muß frei von Seiteneffekten sein

Benchmarks

Overhead von geschützter Kommunikation in Mikrosekunden

Operation	DEC OSF/1	Mach 3.0	**SPIN**
Protected in-kernel call	nicht verfügbar	nicht verfügbar	.13
System-call	5	7	4
Cross-address-space-call	845	104	89

Overhead des Thread-Managements in Mikrosekunden

Operation	**DEC OSF/1**		**Mach**		**SPIN**		
	kernel	user	kernel	user	kernel	user	
						layered	integrated
Fork-Join	198	1230	101	338	22	262	111
Ping-Pong	21	264	71	115	17	159	85

Zusammenfassung

SPIN

- **Erweiterbarkeit**
- **Sicherheit** — **in einem System**
- **Performance**

- ❑ **Events vereinen flexible Erweiterbarkeit mit der Einfachheit von Mechanismen auf Sprachebene**
- ❑ **Erweiterbarkeit durch Event-Mechanismus schnell und sicher**