Taskservice

Stefan Brähler, Martin Kiefel

SDI2

June 4, 2009

Vortragsübersicht

- Ziele
- Stand der Dinge
- Erstellen einer Task
- Weitere Funktionalitäten
- IDL4-Interface
- Optionales

Was wollen wir?

- Tasks & Threads erstellen
- Tasks & Threads beenden/löschen
- Kooperation unterstützen
- Information über Threads und Tasks zur Verfügung stellen

Mögliche(s) API

- `ThreadCreate`
- `TaskCreate`
- `Wait`
- `Exit`, `Kill`
- `ListTasks`, etc.

Mögliche(s) API

- `ThreadCreate`
- `TaskCreate`
- `Wait`
- `Exit`, `Kill`
- `ListTasks`, etc.

kleines Beispiel

Um eine fiktive Anwendung “foo” mit zwei Threads zu erhalten:
Shell: `TaskCreate("foo")`, `pager_TaskStart()`
Foo: `ThreadCreate()`, `L4_Start()`

Was wir haben

- `ThreadControl`
- `SpaceControl`
- `L4_Start`

Was wir haben

- `ThreadControl`
- `SpaceControl`
- `L4_Start`

Probleme

- Diese **privilegierten** Syscalls sollten nicht frei zugänglich sein, da sie es ermöglichen Einfluß auf andere Anwendugen zu nehmen
- Ausserdem sollte sich eine Anwedung nicht um die "gory details" kümmern müssen (Fehleranfälligkeit, Modularität, Einfachheit...)
- Accounting wird erst durch zentrale Instanz möglich

ELF Dateien parsen

Erinnerung

Eine ELF Datei besteht aus einem Einsprungspunkt und mehreren Sections mit ihren virtuellen Adressen.

ELF Dateien parsen

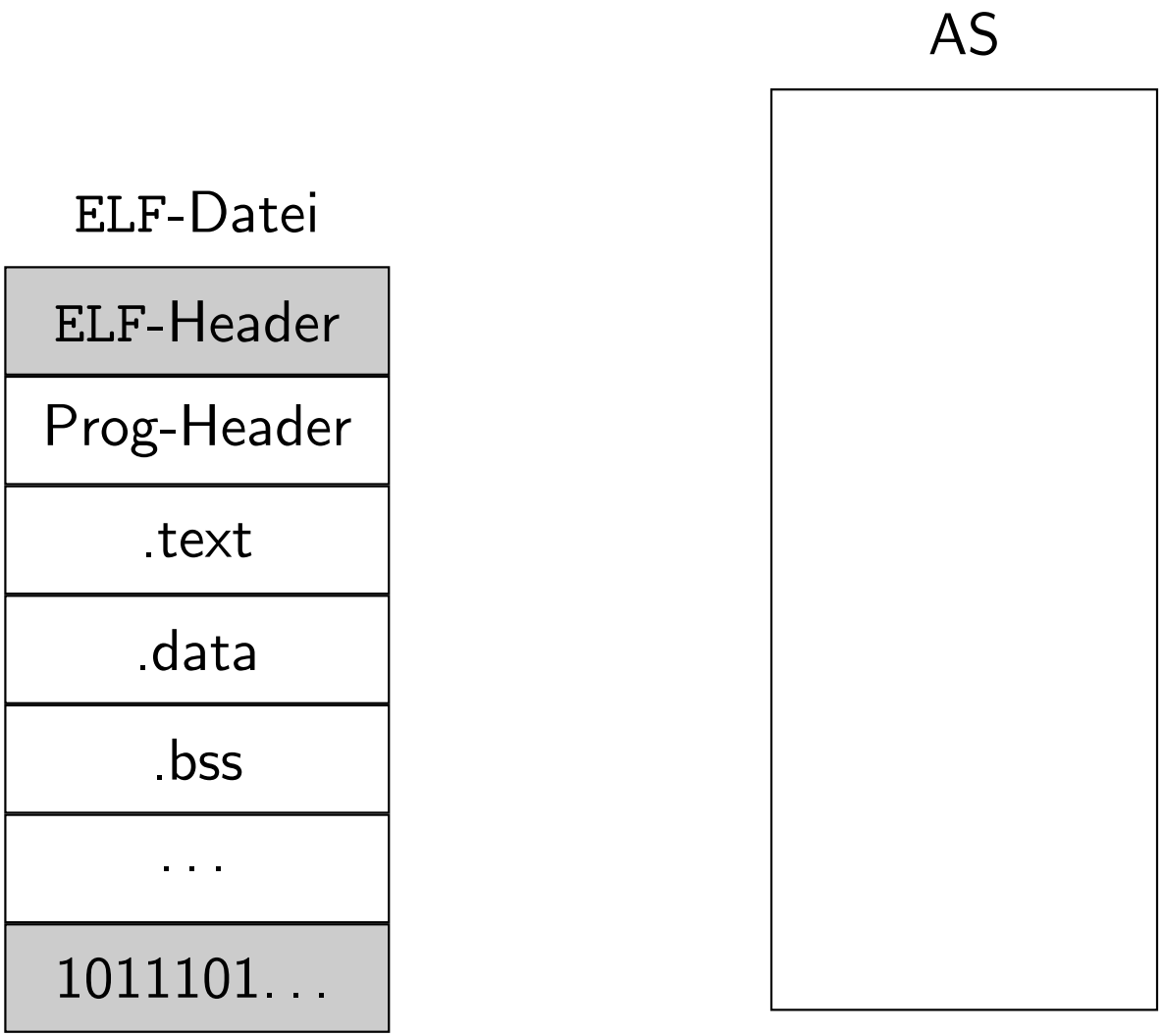
Erinnerung

Eine ELF Datei besteht aus einem Einsprungspunkt und mehreren Sections mit ihren virtuellen Adressen.

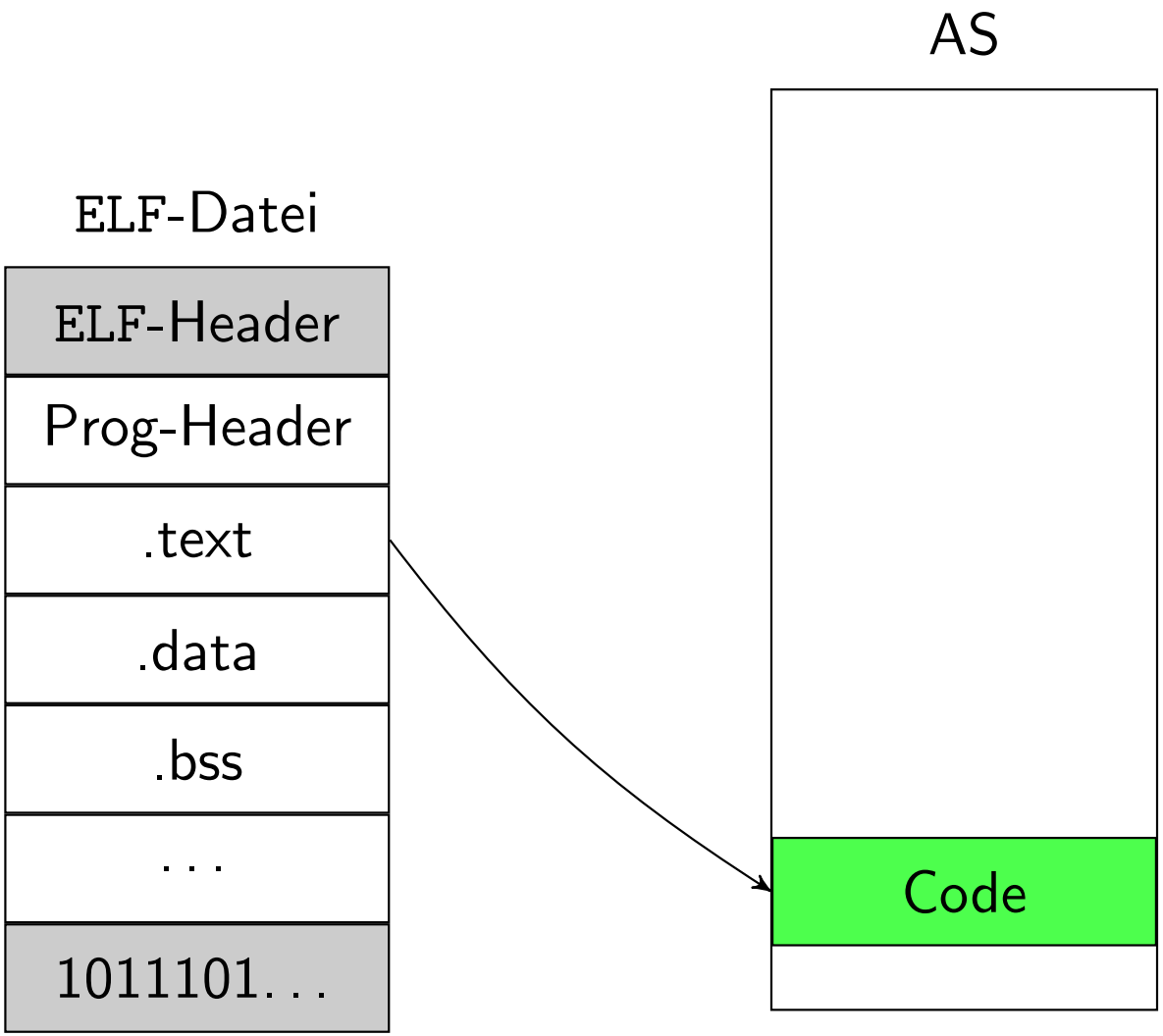
Wo kommt was hin?

Die in den Sections angegebenen virtuellen Adressen geben die Adressen im **ZielAS** an, wohin die einzelnen Sections gemappt werden müssen.

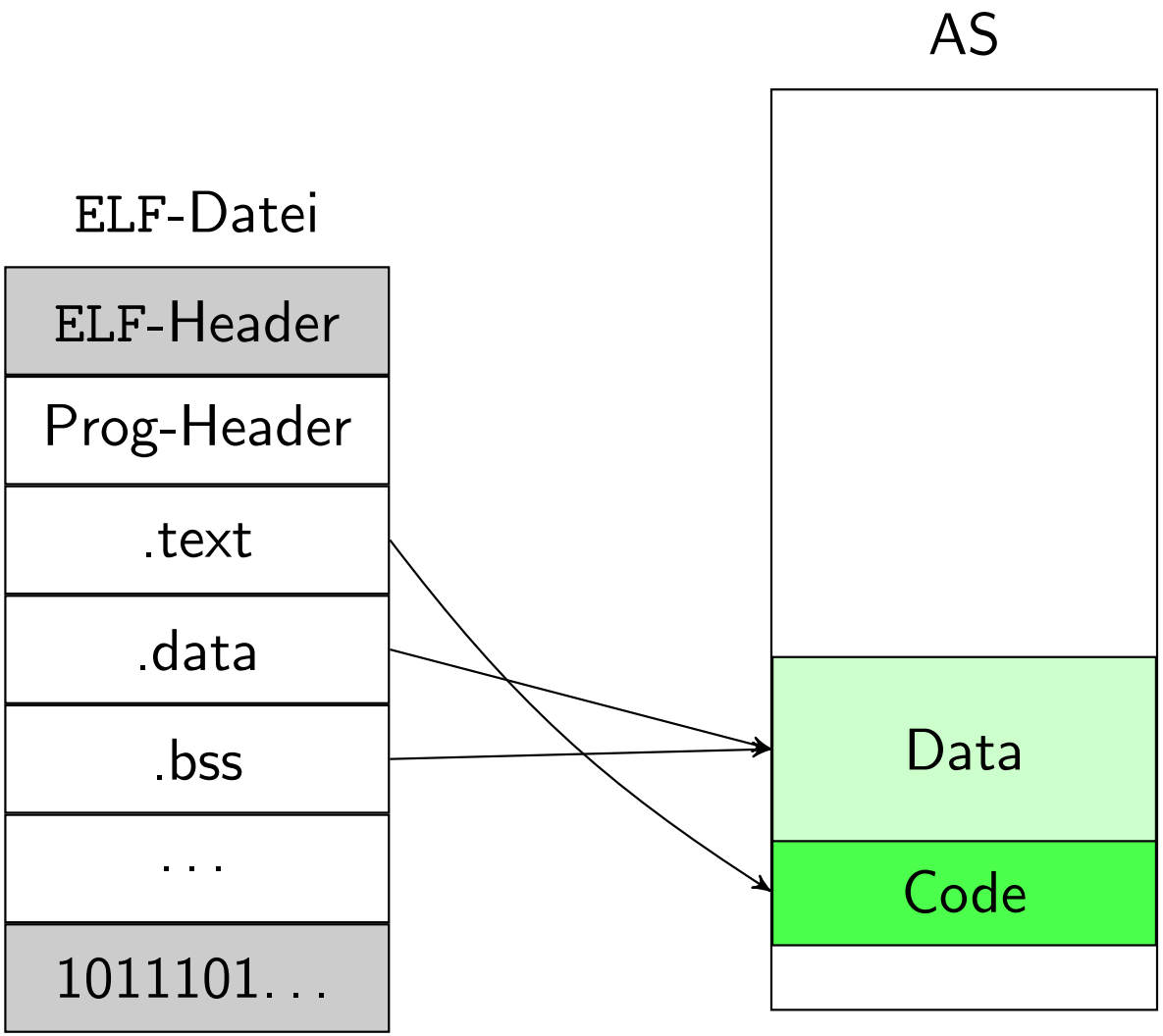
Überführung ELF → ZielAS



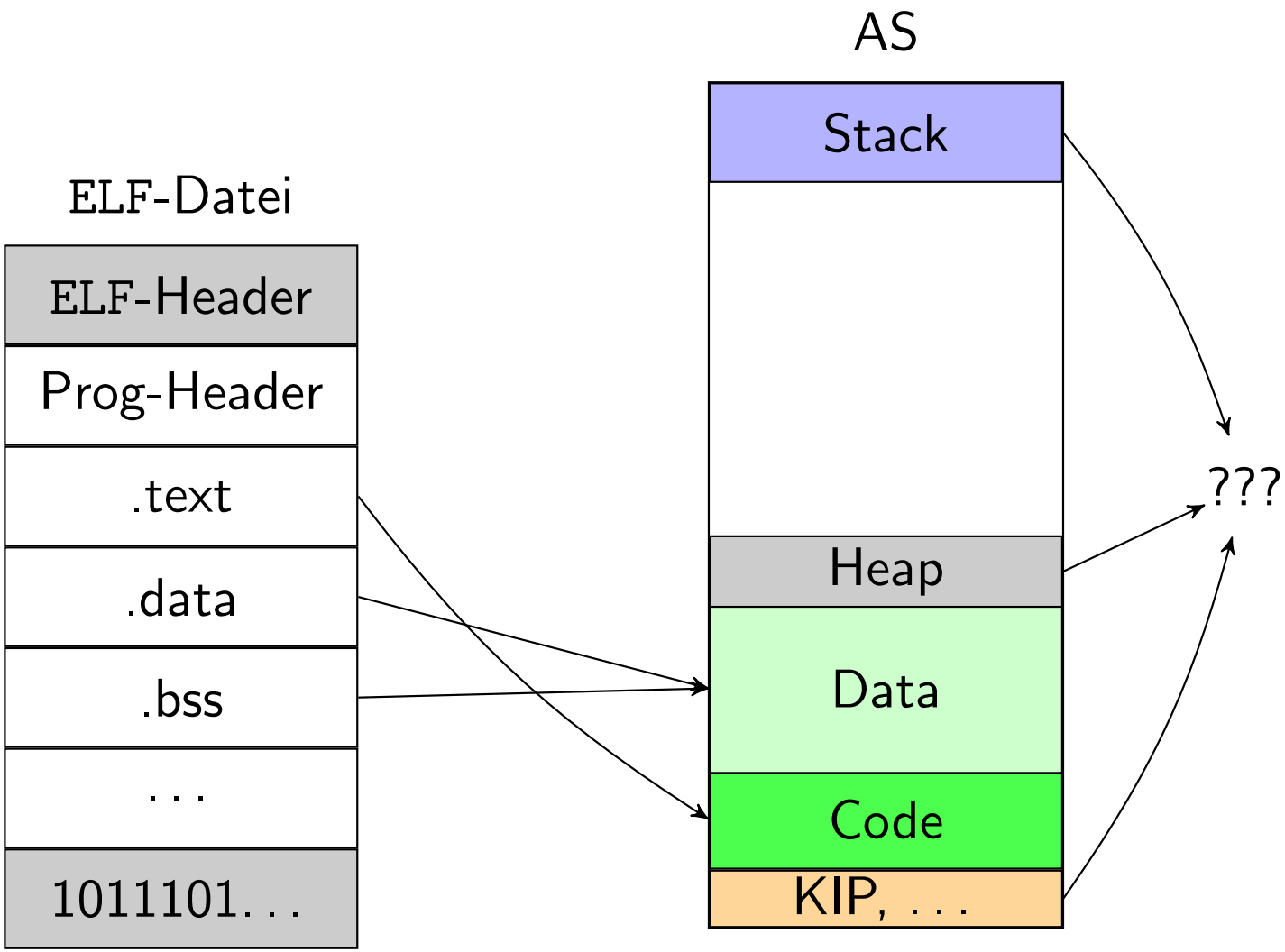
Überführung ELF → ZielAS



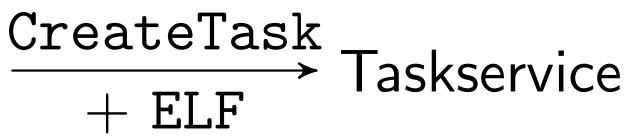
Überführung ELF → ZielAS



Überführung ELF → ZielAS



TaskCreate

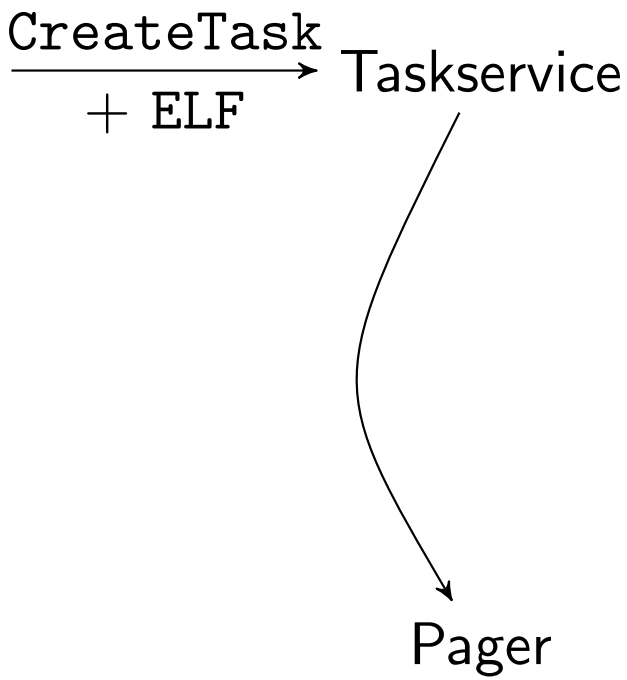


- Eventuell Berechtigungen oder Limits prüfen.
  Neue `ThreadID` wählen.
  Interne Datenstrukturen updaten.

Pager

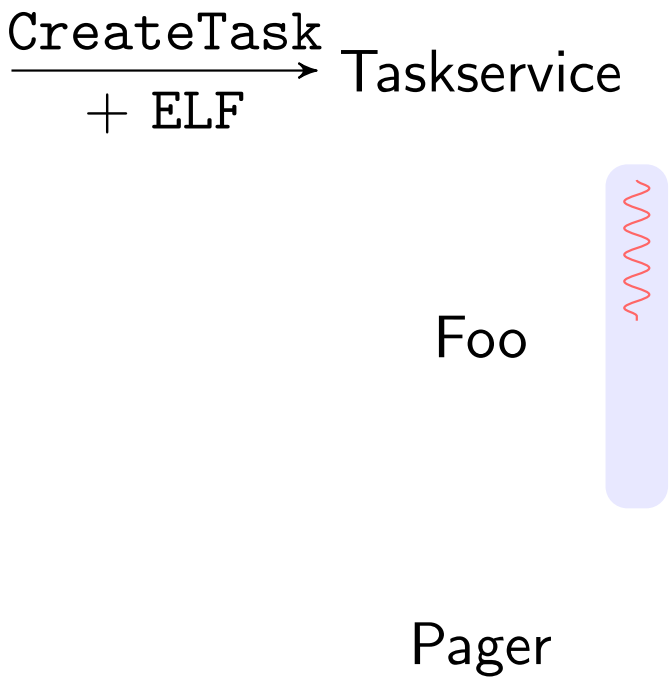
TaskCreate

- Pager/Memoryservice anweisen ELF Datei bei Pagefaults der neuen Task von einem entsprechenden Image zu laden.



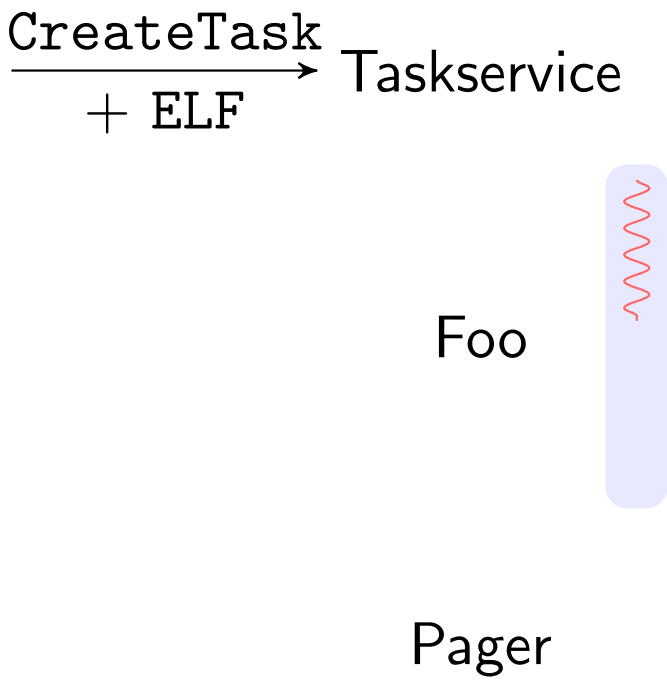
TaskCreate

- Inaktiven Thread + neuem AS mit `ThreadControl` erstellen. Ggf. durch Syscallservice.



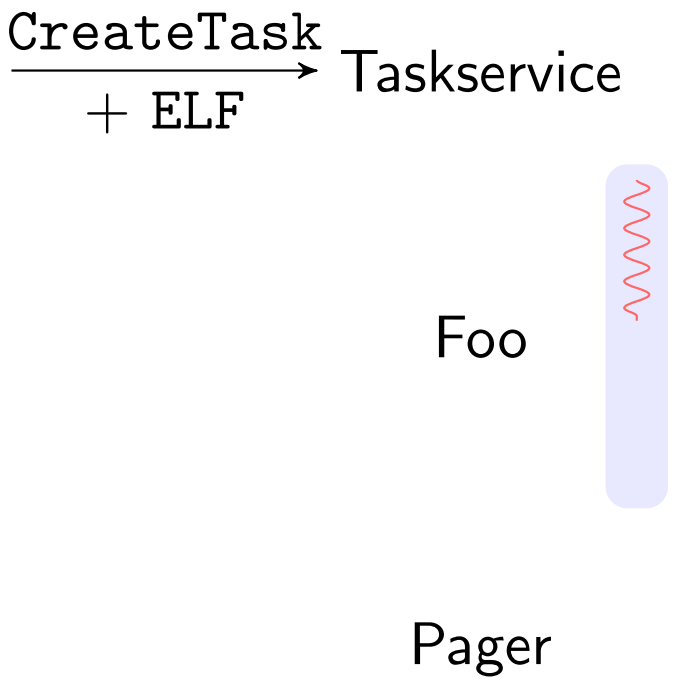
TaskCreate

- UTCB Area und KIP mit `SpaceControl` vom Kernel pinnen lassen.



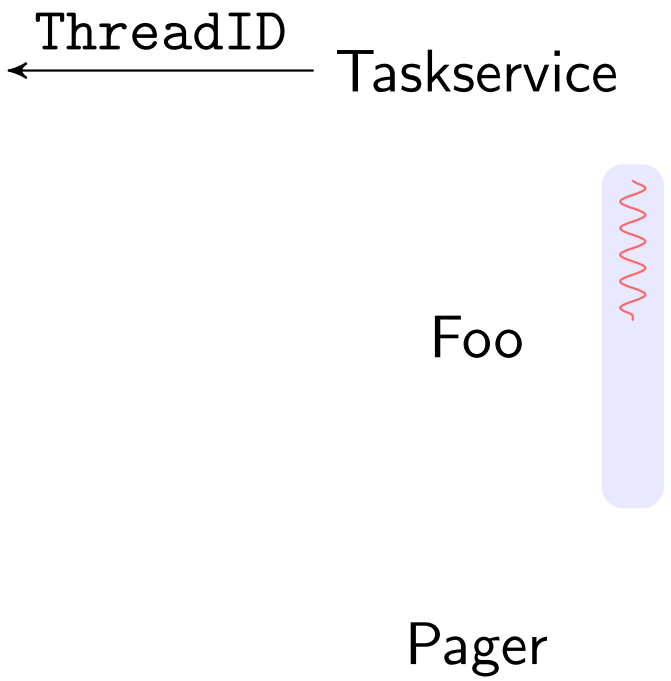
TaskCreate

- Ort des UTCB und Pager durch zweites `ThreadControl` festlegen.



TaskCreate

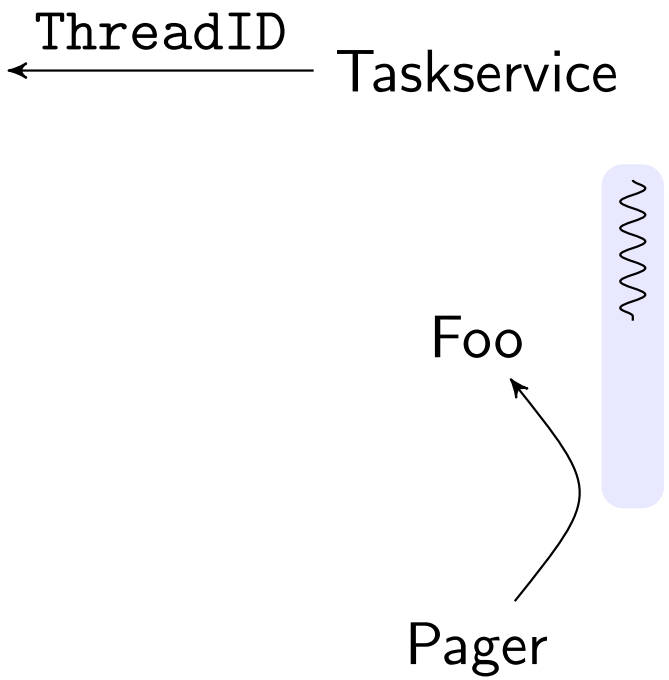
- Neue `ThreadID` an rufenende Anwendung zurückreichen.



TaskCreate

von hier aus: pager_TaskStart

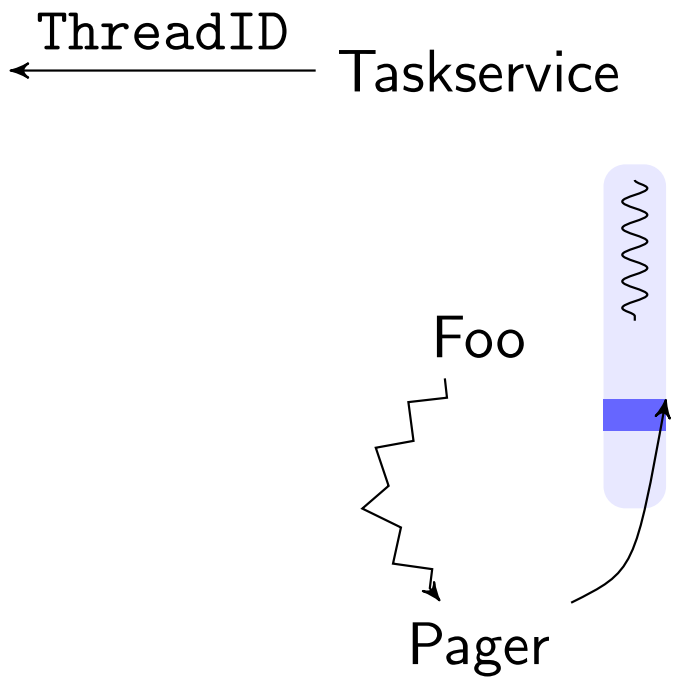
- StartIPC mit IP und SP (IP aus ELF-Datei) an neue Anwendung verschicken.
- Ausgelösten Pagefault mit entsprechendem Mapping beantworten.



TaskCreate

von hier aus: pager_TaskStart

- StartIPC mit IP und SP (IP aus ELF-Datei) an neue Anwendung verschicken.
- Ausgelösten Pagefault mit entsprechendem Mapping beantworten.



ThreadCreate

- Eventuell Berechtigungen oder Limits prüfen.
  Neue `ThreadID` wählen.
  Interne Datenstrukturen updaten.
- Inaktiven Thread (mit Pager und neuem UTCB) in dem AS der rufenden Anwendung erstellen durch `ThreadControl`.
- Pager der Anwendung über neuen Thread informieren und `SP` erfragen.
- `ThreadID` und `SP` an rufende Anwedung zurückgeben.
  (Warum nicht `IP`?)

Kill, Exit & Wait

Kill

Weist Taskservice an Thread mit `ThreadControl` zu löschen.
(Sofern man einen Thread löscht, der alleine einen AS bewohnt, löscht L4 implizit auch den AS)

Kill, Exit & Wait

Kill

Weist Taskservice an Thread mit `ThreadControl` zu löschen. (Sofern man einen Thread löscht, der alleine einen AS bewohnt, löscht L4 implizit auch den AS)

Exit

Ähnlich wie `Kill`, als `ThreadID` wird eigene `ThreadID` verwendet und ein Rückgabewert überreicht.

Kill, Exit & Wait

Kill

Weist Taskservice an Thread mit `ThreadControl` zu löschen.
(Sofern man einen Thread löscht, der alleine einen AS bewohnt, löscht L4 implizit auch den AS)

Exit

Ähnlich wie `Kill`, als `ThreadID` wird eigene `ThreadID` verwendet und ein Rückgabewert überreicht.

Wait

IPC an den Taskservice blockiert so lange bis sich der gewünschte Thread beendet wird.
Der mit `Exit` übergebene Rückgabewert wird ggf. überreicht.

IDL4-Interface

- *TaskCreate*(in path_t file, in arg_seq_t args, out ThreadID tid) raises (permission, oom, filenotfound, io)
- *ThreadCreate*(out ThreadID tid, out long sp) raises (permission, oom)
- *Wait*(in ThreadID tid, out long result) raises (invalidid)
- *Exit*(in long result)
- *Kill*(in ThreadID tid) raises (permission, invalidid)

Optional: Informationen und Verwaltung

Statusinformationen

Attribute (`taskID`, `parent`, `owner`, ...) hinzufügen um z.B.:

- Mittels des Nameservice Interface z.B. /services/task/byTaskID/42/... exportieren
- `TaskID()`, `TaskThreads()` anzubieten

Optional: Informationen und Verwaltung

Statusinformationen

Attribute (`taskID`, `parent`, `owner`, ...) hinzufügen um z.B.:

- Mittels des Nameservice Interface z.B. /services/task/byTaskID/42/... exportieren
- `TaskID()`, `TaskThreads()` anzubieten

Verwaltung

Globale/lokale Limits/Policies durchsetzen, z.B.:

- Anzahl Threads pro Task
- Tasks pro User
- MaxTasks im System
- Accounting (benötigt u.a. Informationen vom Scheduler)

Fragen?