# Dynamic updates for object-oriented operating-system kernels

Tobias Langer

30.04.14

# Motivation

- Rising complexity of modern software
- Need for early updates and patches
- High availability demands on software
- Restarts must be avoided where possible

### Solution: Dynamic Update

  - Install software updates at runtime
  - Update gets effective without restart

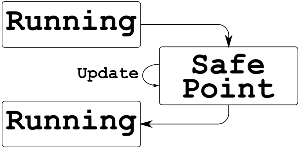
# Dynamic Update

## Dynamic updates must do two things:

- Update of the code section
- Transfer of the current state

## Basic flow of a dynamic update is:

- Application of update at safe point
- Transfer of state information
- Invocation redirection



# ...in operating systems

- Operating systems must fulfill the same requirements
- ...but impose further requirements to the update system
  - No or limited runtime system
  - ...

For dynamic update support, operating system must offer:

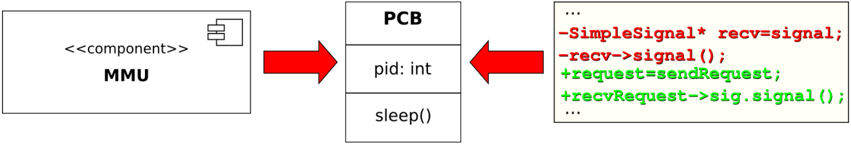
- Updatable unit
- Safe point
- State tracking
- State transfer
- Redirection of invocations
- (Version management)

# Object-oriented dynamic update

- Objects contain state information and code
- Compromise between modules and binary-rewriting

## Downsides:

- Updatable code is bound to system layout
- Dependency of dynamically resovled references
- Makes many optimization impossible



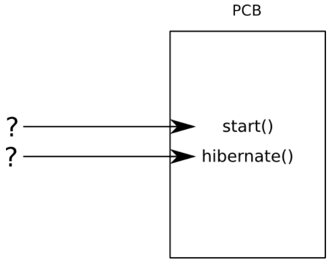
# Implementation with object-orientation

## Dynamic update with object-orientation

- Updatable unit $\rightarrow$ Objects
- Safe point $\rightarrow$ ?
- State tracking $\rightarrow$ List of instances
- State transfer $\rightarrow$ Object replacement
- (Version management)

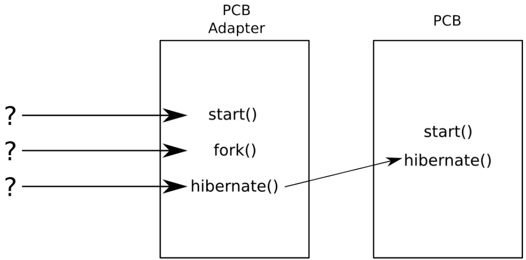
# Interposition

- Hiding an object behind another object
  - Forward calls to the object with addtional prologues / epilogues
  - Interface changes for the callers view
  - Provide new implementations of the objects methods
- Realized via the Adapter pattern



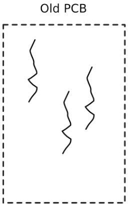
# Interposition

- Hiding an object behind another object
  - Forward calls to the object with addtional prologues / epilogues
  - Interface changes for the callers view
  - Provide new implementations of the objects methods
- Realized via the Adapter pattern



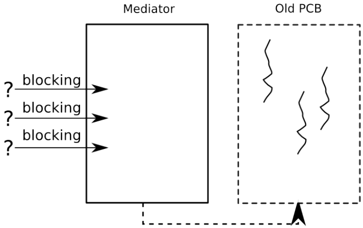
# Hot-Swapping

- State tracking problem
- References might be in use, when object is switched
- Interpose mediator object
  - Keeps track of currently used instances
  - Blocks all new calls to the object
  - When quiescence is reached, object is replaced



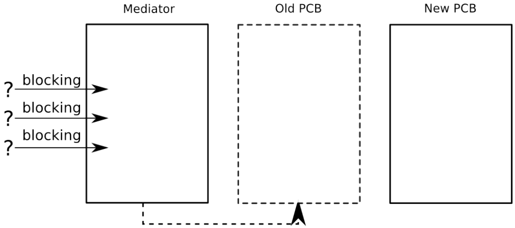
# Hot-Swapping

- State tracking problem
- References might be in use, when object is switched
- Interpose mediator object
  - Keeps track of currently used instances
  - Blocks all new calls to the object
  - When quiescence is reached, object is replaced



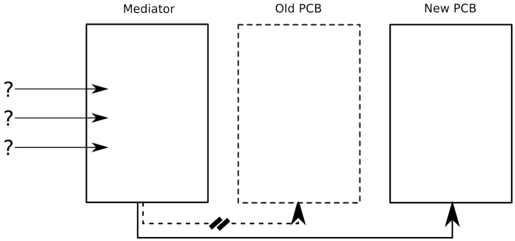
# Hot-Swapping

- State tracking problem
- References might be in use, when object is switched
- Interpose mediator object
  - Keeps track of currently used instances
  - Blocks all new calls to the object
  - When quiescence is reached, object is replaced



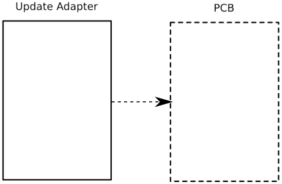
# Hot-Swapping

- State tracking problem
- References might be in use, when object is switched
- Interpose mediator object
  - Keeps track of currently used instances
  - Blocks all new calls to the object
  - When quiescence is reached, object is replaced



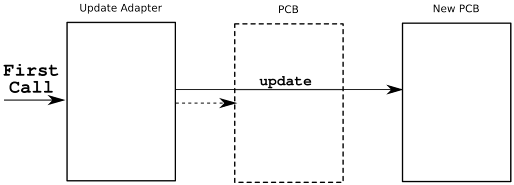
# Lazy Update

- Update of all instances takes time
- Not all instances need to be updated
- Mark them for update, update on first following call
- Future calls go to the updated object



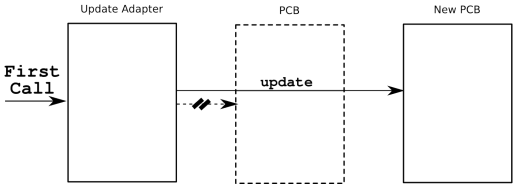
# Lazy Update

- Update of all instances takes time
- Not all instances need to be updated
- Mark them for update, update on first following call
- Future calls go to the updated object



# Lazy Update

- Update of all instances takes time
- Not all instances need to be updated
- Mark them for update, update on first following call
- Future calls go to the updated object



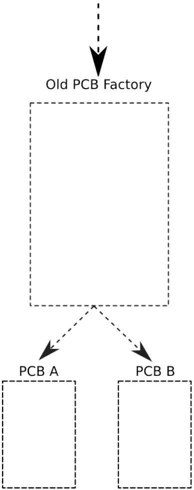
# K42 Introduction

- Research kernel developed by IBM
- Focus on Linux API and ABI support
- Strongly modularized by application of object orientation
- Event-driven with short-lived kernel threads

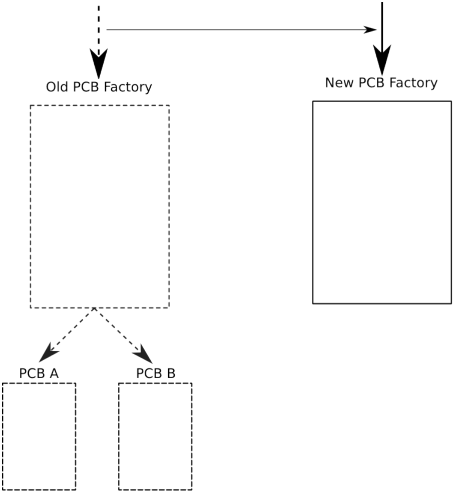
# Dynamic update in K42

- Factories for all objects
- Factories keep track of all created instances
- Dynamic update results in:
  1. Hot-swapping of the factory
  2. (Lazy) replacement of the existing instances

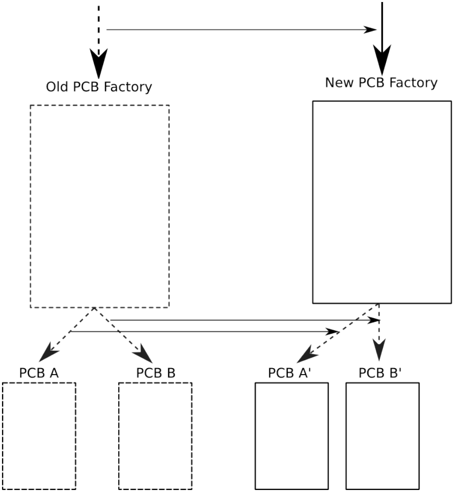
# K42-Style Factories



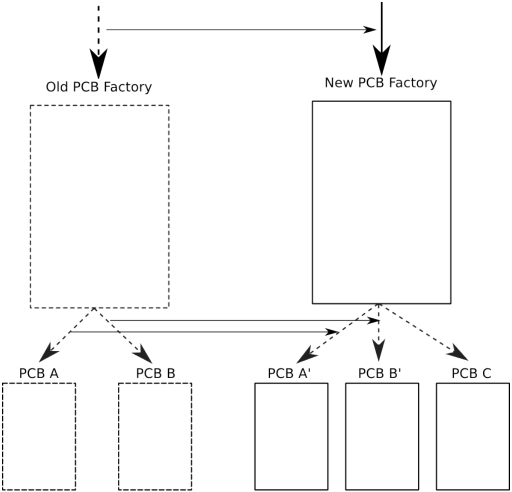
# K42-Style Factories



# K42-Style Factories



# K42-Style Factories



# Quiescence detection

Problem: How find safe point

- K42 Threads are short lived
- Exist in generations
- Updating sets a new generation
- Wait for all threads of last generation to complete
- Then update the object

# Discussion

## Dynamic update system is based on dynamically resolved references

- Forbids optimizations
- Severe as objects are on a fine-grained level
- Comparision of two running systems would be interesting... but is not given :(

## Upsides:

- Interesting approach
- Extendable for well modularized operating systems (Linux,...)
- ...and also for application software