



Hardware / Software Integration through Modeling and Automatic Code Generation

Frank Weil
Manager, GSG Software Design Automation

Thomas Weigert
Senior Director, GSG SE

Overview

**A common platform can
reduce development costs,
but it is not sufficient in itself**

Applications are integrated into platforms

Interfacing application code to development platforms ...

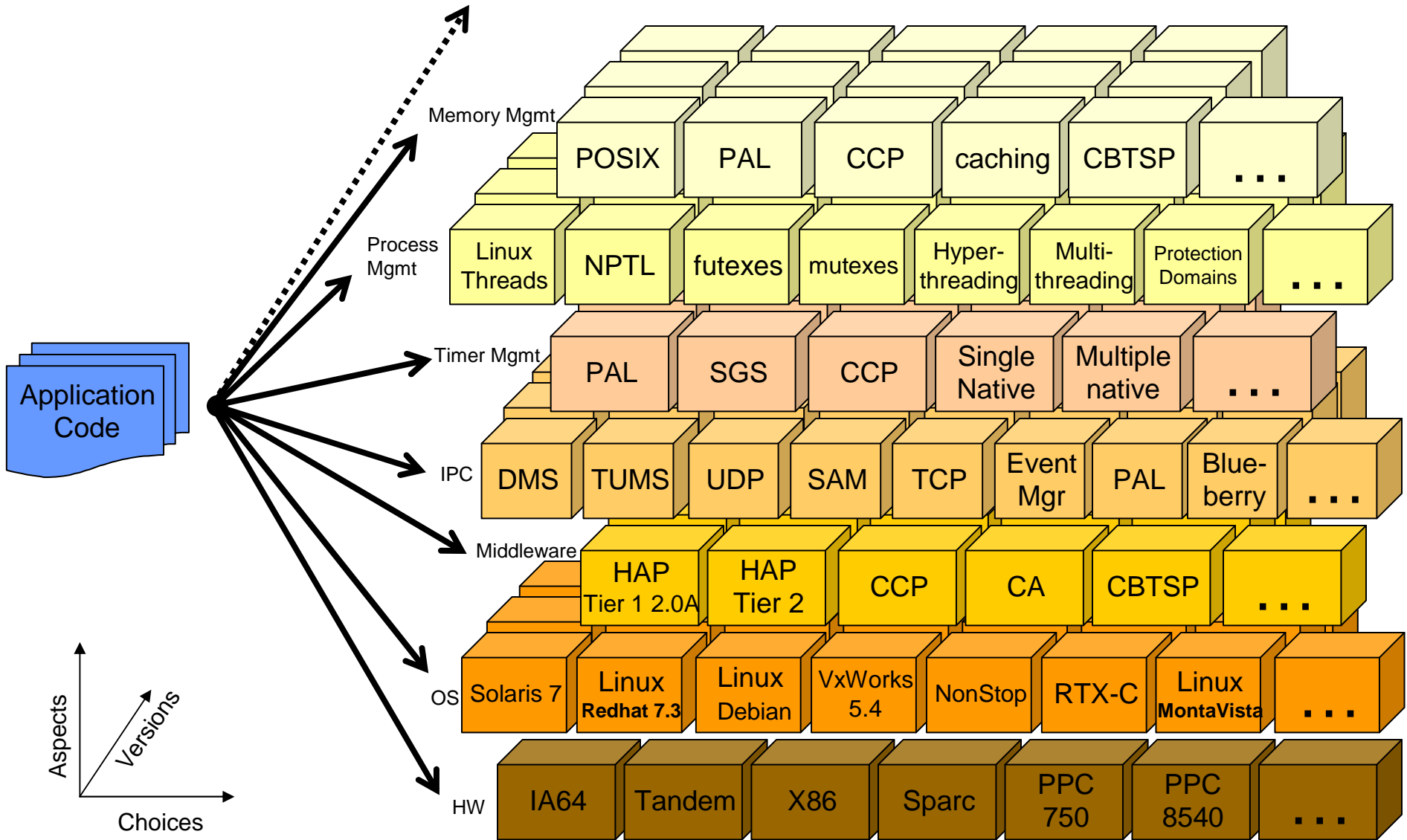
- Is complex
- Is error prone
- Requires expertise beyond the product functionality
- Often changes over the life of the product

A “common platform” is really composed of many components

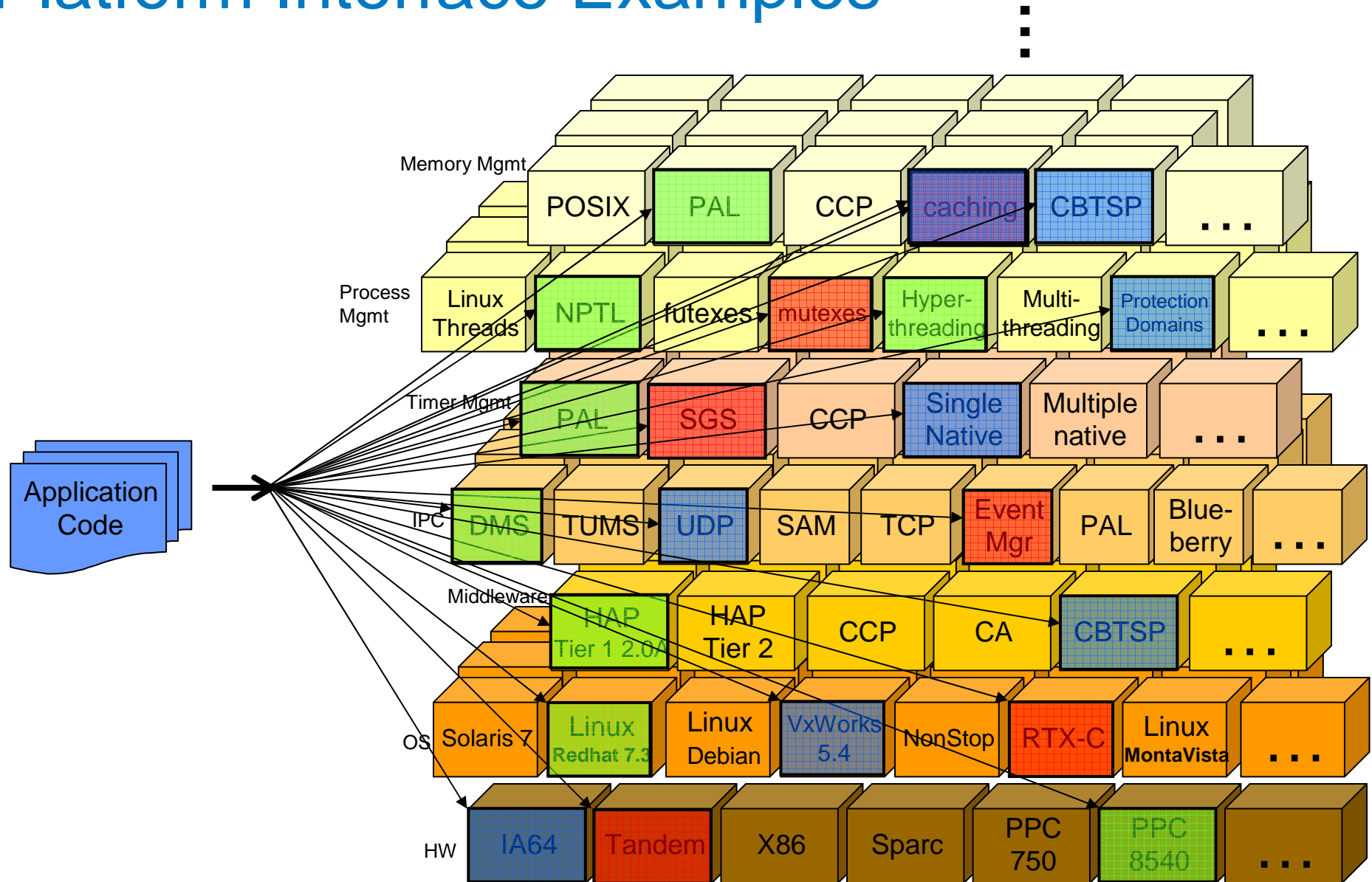
- Multiple layers of functionality
- Many versions of the components

Platform Details

⋮



Platform Interface Examples



Separate Development and Platform

Two aspects

- Create designs independent of platforms
- Automatically target the resulting code to the platform

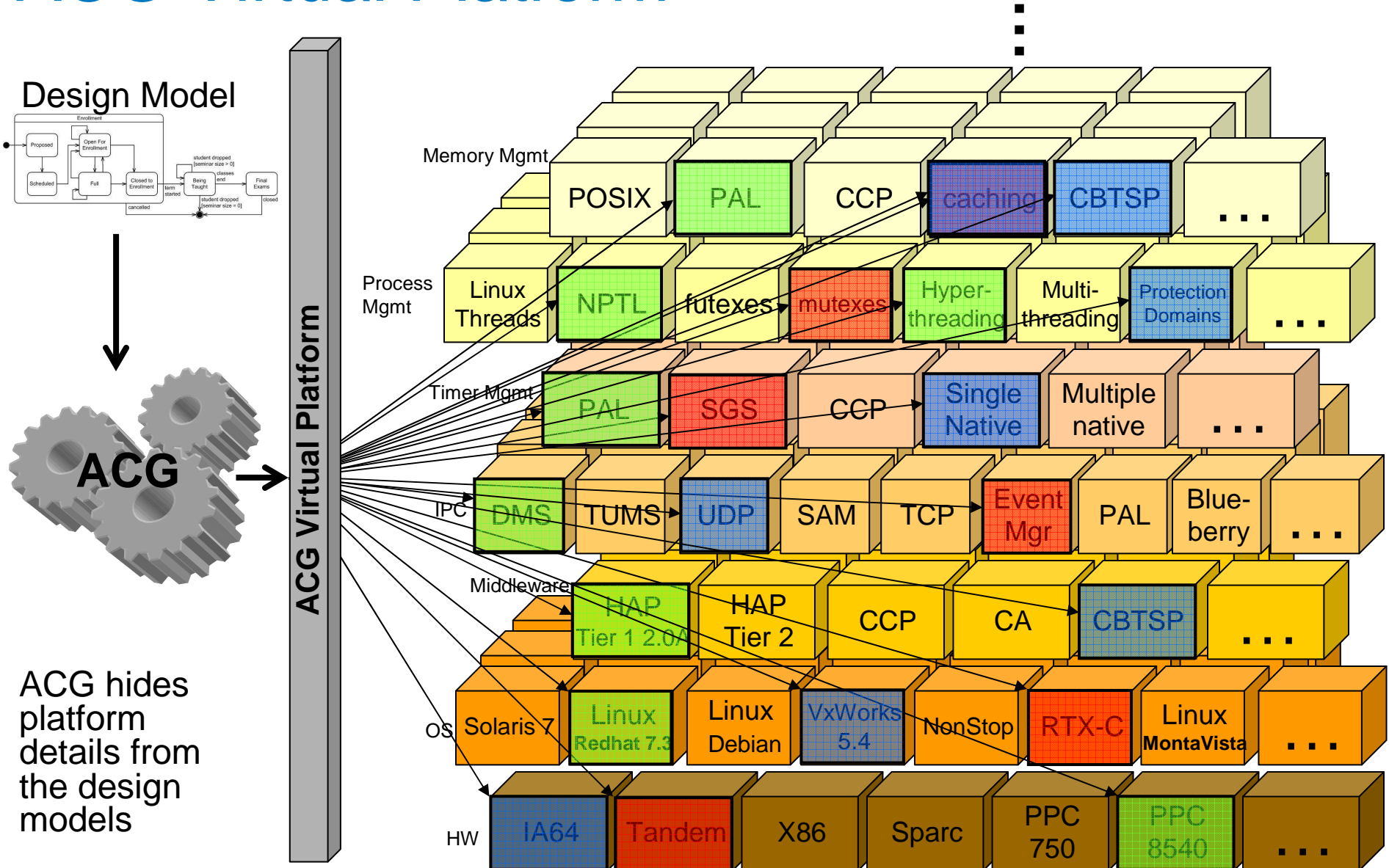
Abstract Design Models

- Capture the functionality without capturing the platform details
- Allow a suitable separation of concerns: product vs. platform

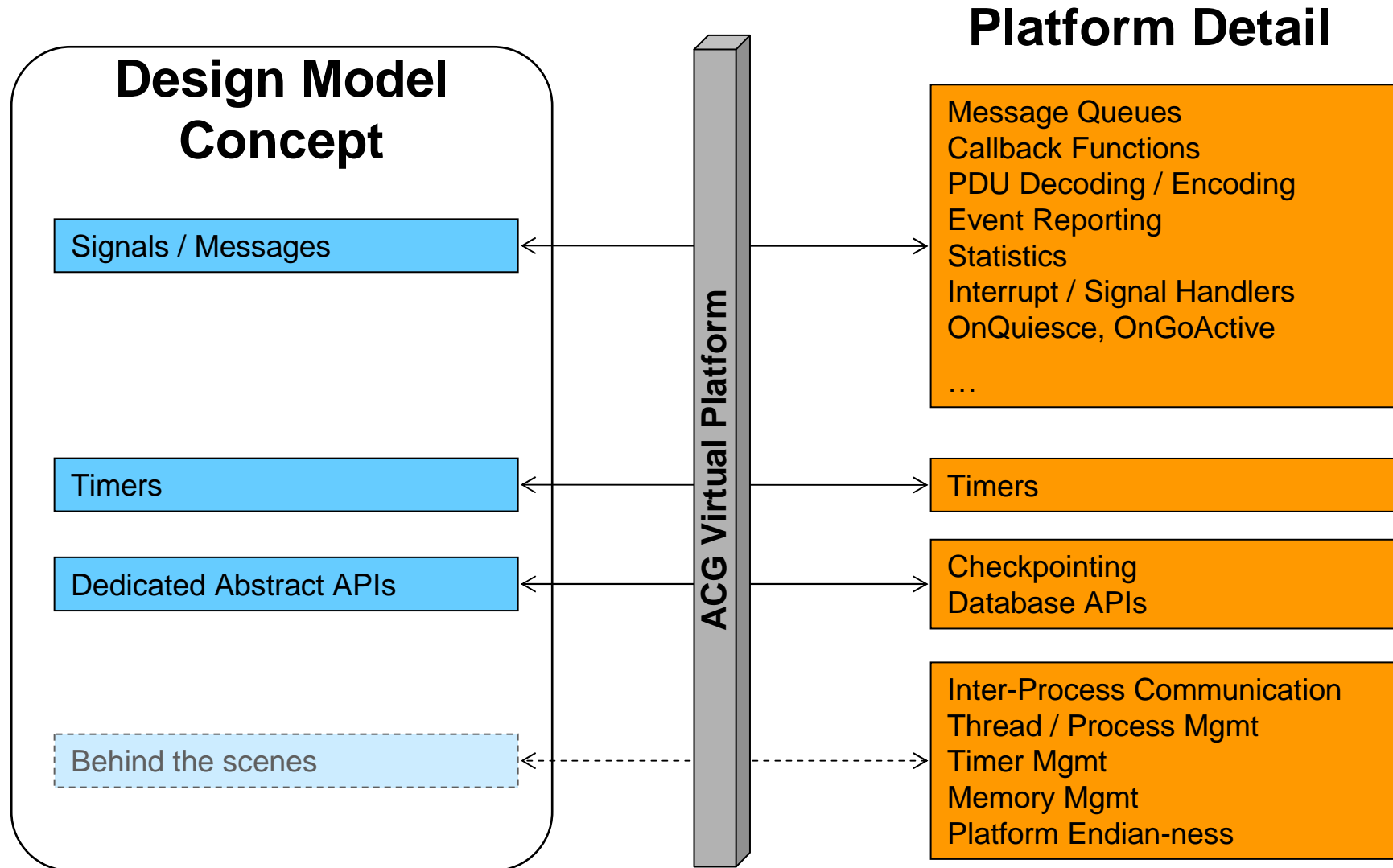
Automatic Code Generation (ACG)

- Hides the details of the underlying platform
- Can target the same abstract design model to multiple platforms
- Creates a virtual common platform

ACG Virtual Platform



Mapping Platforms to Applications



Examples

Base Site Controller

Moved application from a rack of GPROC cards (MC6809 cards with distributed memory) to a shared-memory Tandem Puma machine.

All changes required to the software architecture were automated by the code generator.

Node B

Changed platform transparently to the application to avoid deadlock problems which had appeared in the hand-written code.

All changes required to the software architecture were automated by the code generator.

Porting effort was reduced from 80 staff days to 10 staff days.

Benefits

- **Designs are easier to produce**
- **Designs are more abstract**
- **Designs are independent of the target platform**
- **Enables rapid retargeting to different platforms**
- **Platform expertise is separate from domain expertise**
- **Common designs can be kept for multiple platforms**

Modeling and Automatic Code Generation
abstract the functionality
to provide a virtual platform.

Platform-specific details
are kept out of the designs

Challenges

Common, standardized APIs should be defined

- E.g.: What are “time” and “timers”?
- Current middleware APIs are programmer-friendly, *not* code-generation-friendly

Can one completely ignore the target platform in the model?

- System functionality may target platform features
- How do you handle this slippery slope?

Platform targeting *may* be approachable as an Aspect

- What are the characteristics that are captured as Aspects?
- How do you define appropriate join points for models?