

PHILIPS

sense and simplicity

Digital Television With Linux

Architecture and Opportunities

Bas Engel

Philips Consumer Lifestyle - Business Unit TV

ELC Ede, November 6, 2008

Contents Of Presentation

- What is Digital TV?
 - TV domain and driving factors
 - Linux and the architectural challenge
- SPACE
 - Architecture and concept
 - Experiences in bringing it towards a product
- Next steps
 - Dynamic application integration
 - Opportunities to improve system integrity

What is digital TV?
TV domain and architectural challenge

Digital TV?

- TV is far away from being just a large screen with a tuner
 - Though it still has the large screen and the tuner
- There are many similarities to other domains
 - Digital STB is a TV without a screen
 - PC can do TV watching and more
 - Phone get's more connectivity features like TV
- Consequently the TV domain is continuously changing
 - And we need to add Philips specific innovations as well
 - In an ever shorter lead time

The Digital Broadcast Domain

- STB drives a huge amount of operator diversity using variations of the standards
 - To differentiate offering
 - No strive towards standardization
 - Country specific
- Also public operators are becoming increasingly diverse
 - To differentiate offering
 - Sometimes city specific
- An increasing amount of diversity in software is required to support this
 - With subtle but fundamental differences

Standards



Applications



Devices



The Connectivity Domain

- There is an increasing amount of content available in the household
 - And an increasing amount of standards to access it
- We are used to a wide set of interactive applications to access the content
 - And hence got used to it
- Most electronic devices nowadays offer ways to access the content
 - Some work better than the other

Standards



Applications



Devices



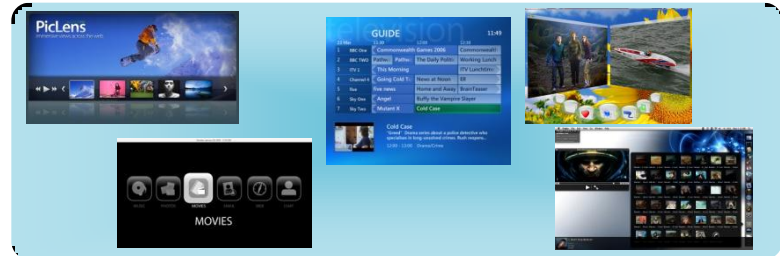
The User Experience Domain

- Aesthetics essential part in user experience
 - Graphical experience is first view
 - A wide set of standards that provide ways to enhance the aesthetics
- User experience is about managing your content independent where it is located
 - Many reference applications
 - Widely used in the internet domain
- Ongoing reference set in the market
 - From the mobile, STB, and PC domain

Standards



Applications



Devices



The Traditional TV Domain

- Top notch picture quality
 - Ongoing drive in the market to improve picture quality
 - Focus on all content areas (analog, digital)
- Ambilight as picture quality extension
 - Improve Ambilight to be a full extensions of TV image
- This is what differentiates a TV from all other devices
 - Continues to be the big screen in the center of the house hold
 - Increasingly bring you the content you want

Picture Quality

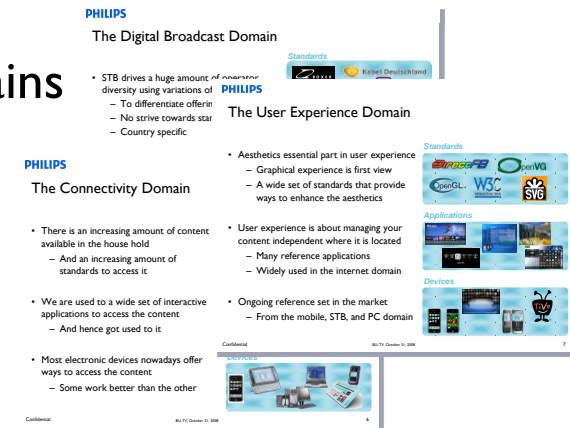


AmbiLight

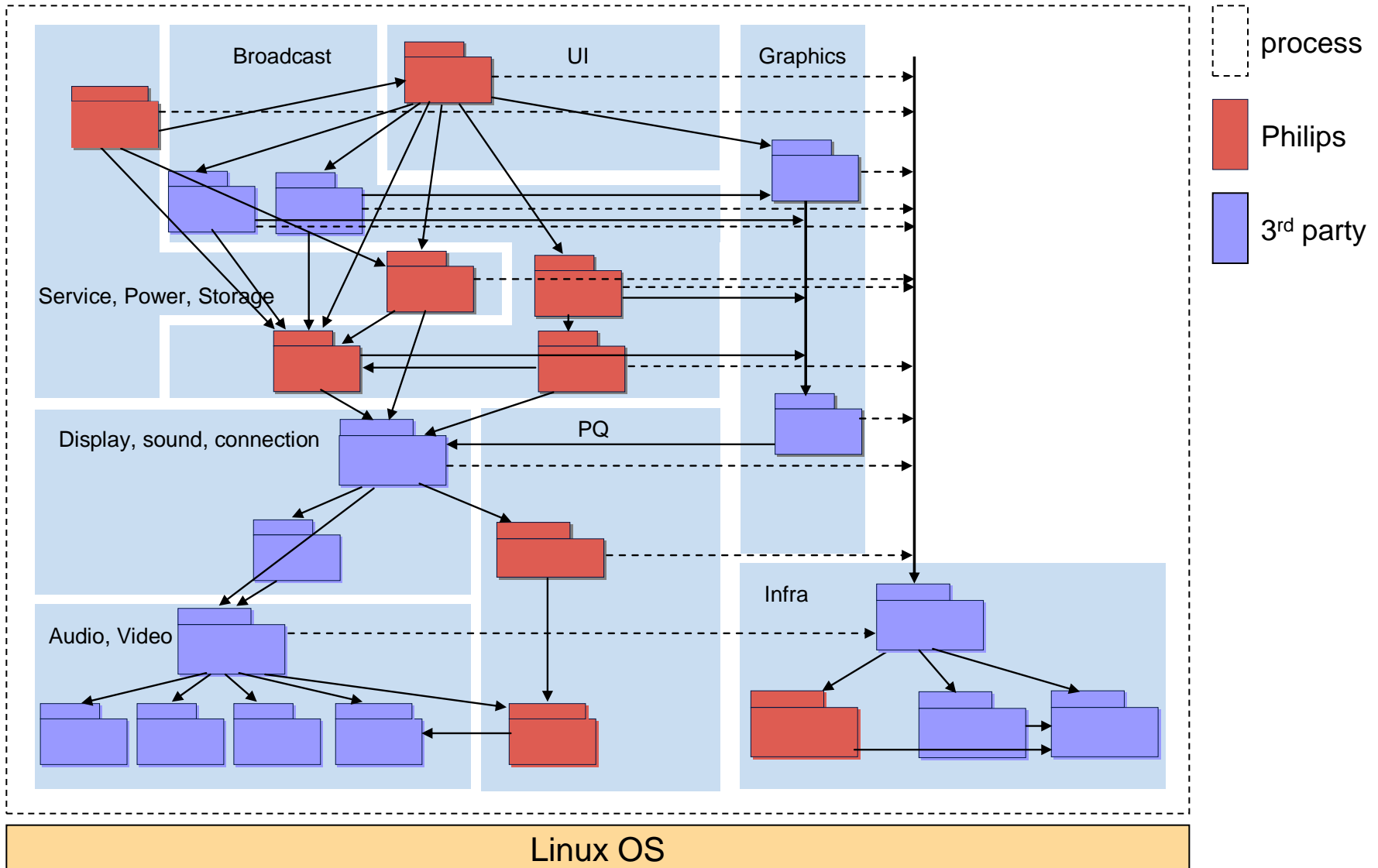


What Will We Explain Today?

- Standards and applications are driven from other domains
 - Wide range of existing solutions in the market
 - Both commercially and open source
 - Increasing attention for connectivity and UI
- Increasing amount of attention for CE industry in community
 - Linux offered by most silicon vendors
 - DirectFB widely picked up as standard graphics approach
 - Resource limitation is no longer a constraint, but embedded in the design
- However, making a CE product out of it is something else
 - Integrating heterogeneous SW is an increasing challenge
 - Heterogeneous being SW from different solutions and domains



Traditional SW Approach: Integrate

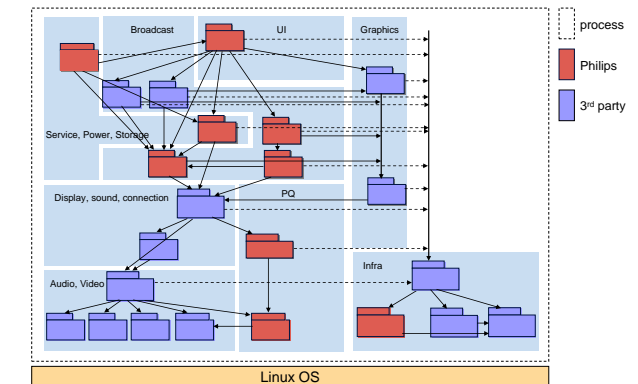


Integration Nightmare

- Integration has it's advantages
 - Optimized resource usage
 - Standard architecture approach (training, integration, etc)

- Integration pitfall
 - High effort due to increased complexity
 - Longer TTM as there is no small change
 - System validation requires global big bang approach
 - No independent lifecycle, no distributed integration cycle

Traditional SW Approach: Integrate



Confidential

Philips Consumer Lifestyle - Business Unit TV, Bas Engel, November 6, 2008

10

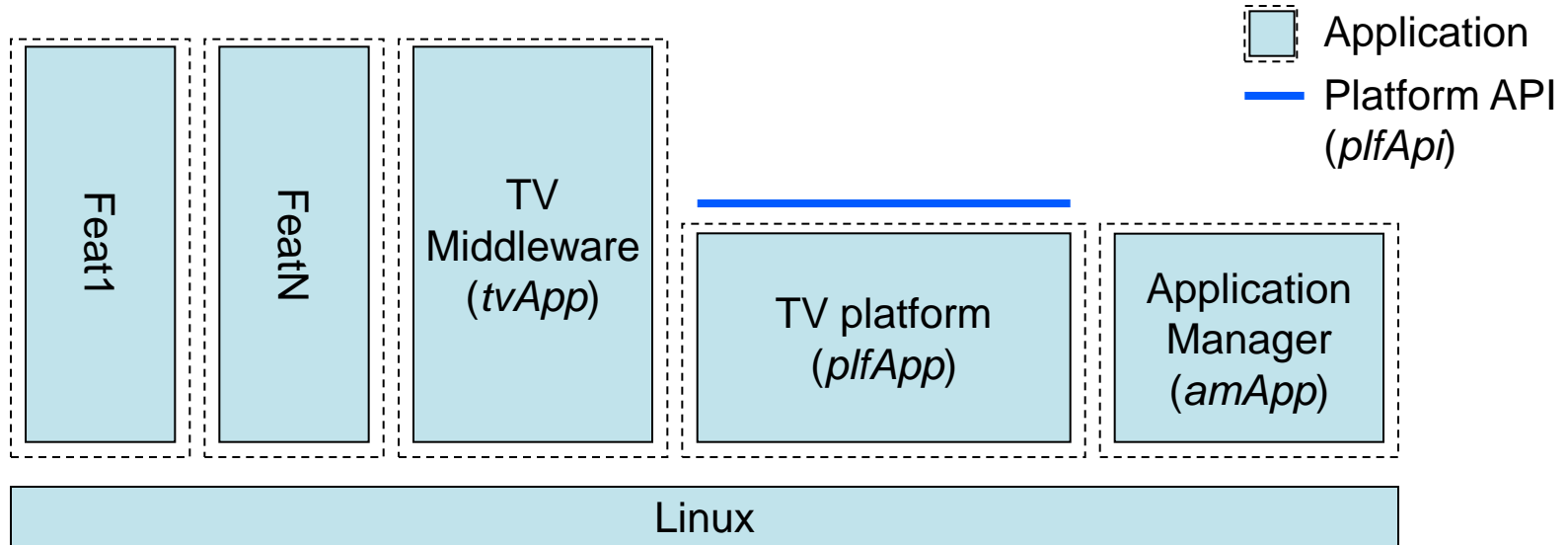
What We Need: Agile Integration Architecture

- Fast and predictable integration of system extensions
 - Avoid an extensive (re)validation cycle
 - Enable convincing module test opportunities
 - Enable PC based testing
- Cater for sharing of critical system resources
 - Audio, Video, and Graphics
 - General purpose (Linux) infrastructure
- Manage building blocks fully independently
 - Limited building block correlation
 - Cater for extensions without the need to know all the details
 - Independent application lifecycle and execution behavior

SPACE

Architecture and concept

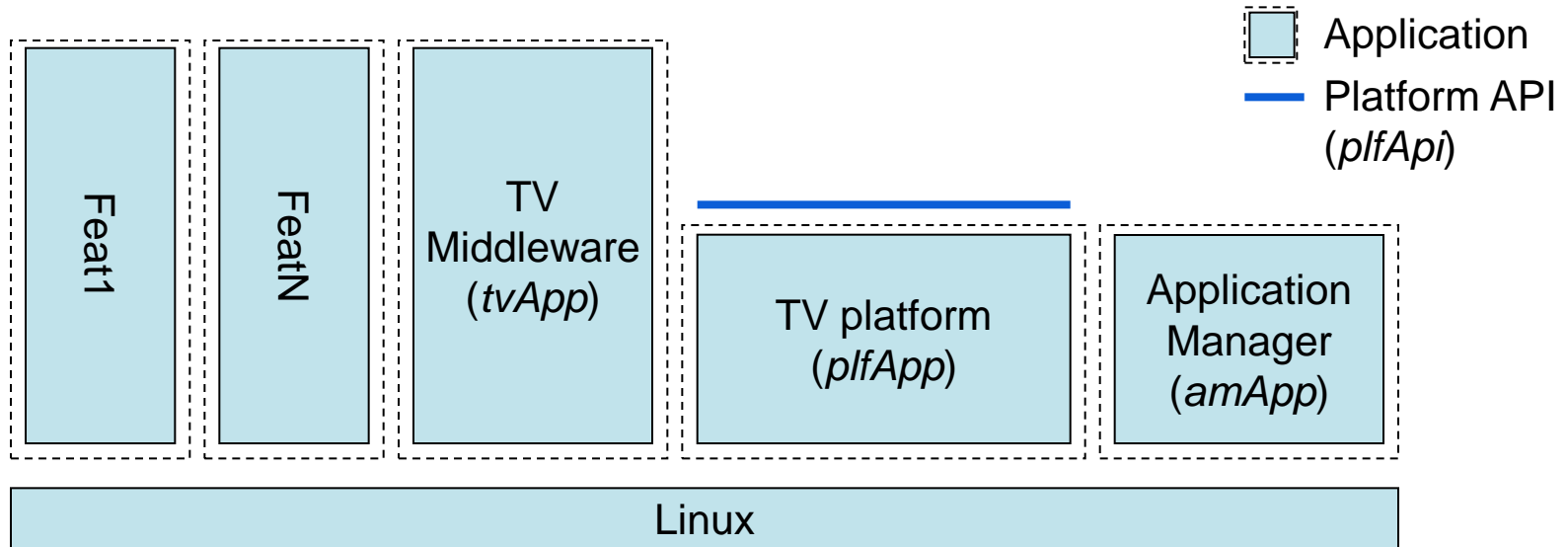
Introducing SPACE



SPlit Application arChitecturE

- Applications are isolated in dedicated processes
- The resources in the system are explicitly and centrally managed
- The client applications are system context unaware
- The lifecycle, focus and visual layout of the client applications is centrally managed

Key Architecture Rule



pAPI only allowed IPC

- Synchronous function call allowed only towards *plfAp*
 - Gated via resource model
- Notifications/events are allowed between applications
 - Without any synchronization allowed

Applications orthogonality

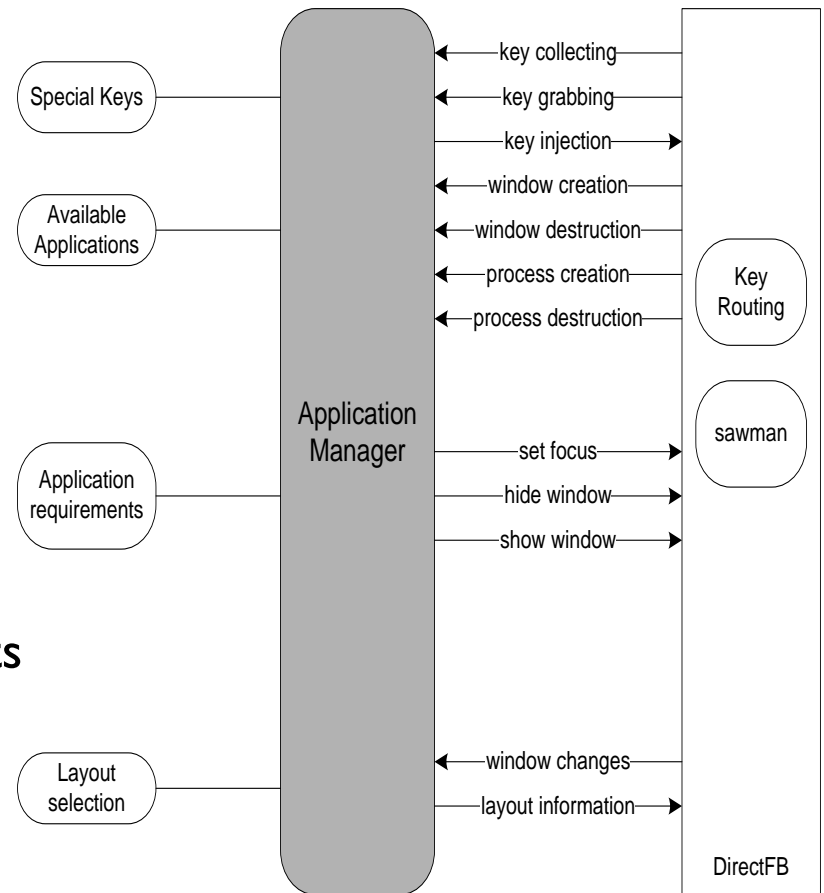
Dynamic behavior cannot be dependent on other applications

Essential Building Blocks

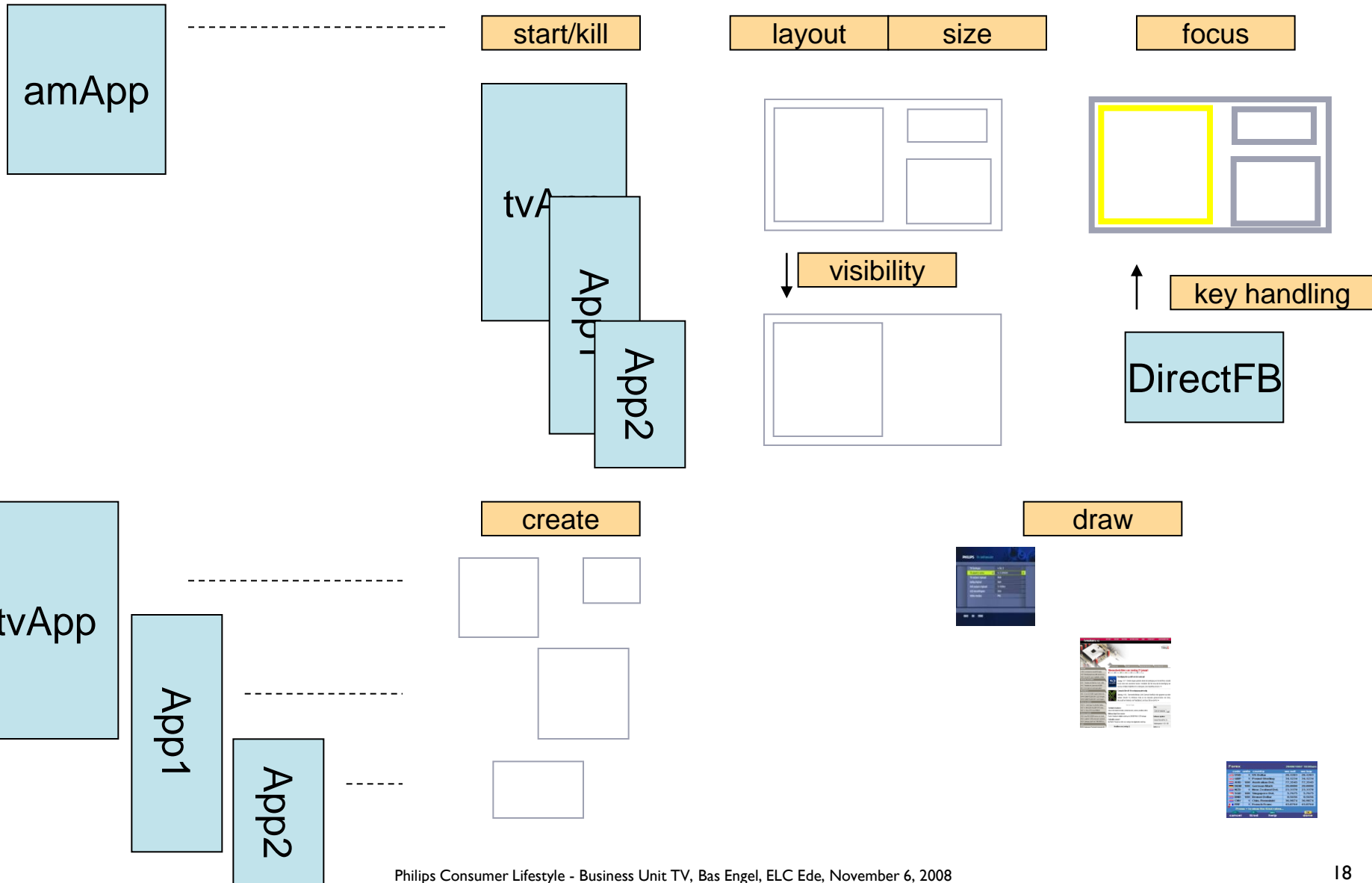
- Application Manager (amApp)
 - Control center of the system
 - Manages application lifecycle, focus, and layout
- Television Application (tvApp)
 - Core TV functionality
 - Broadcast, installation, and general TV settings
- Platform Application (plfApp)
 - HW platform abstraction layer
 - Audio Video and Graphics control
- Platform API (plfApi)
 - Philips AVG control interface
 - Philips Audio-Video API to control platform by applications

Application Manager

- Application lifecycle management
 - Starting, stopping applications based on remote control keys
 - Knows the system requirements of all applications
- Focus handling
 - Determines the active window, reacts to user request
 - Manages the application requirements to the AV resources
- Layout management
 - Determines how the applications are presented



Responsibilities



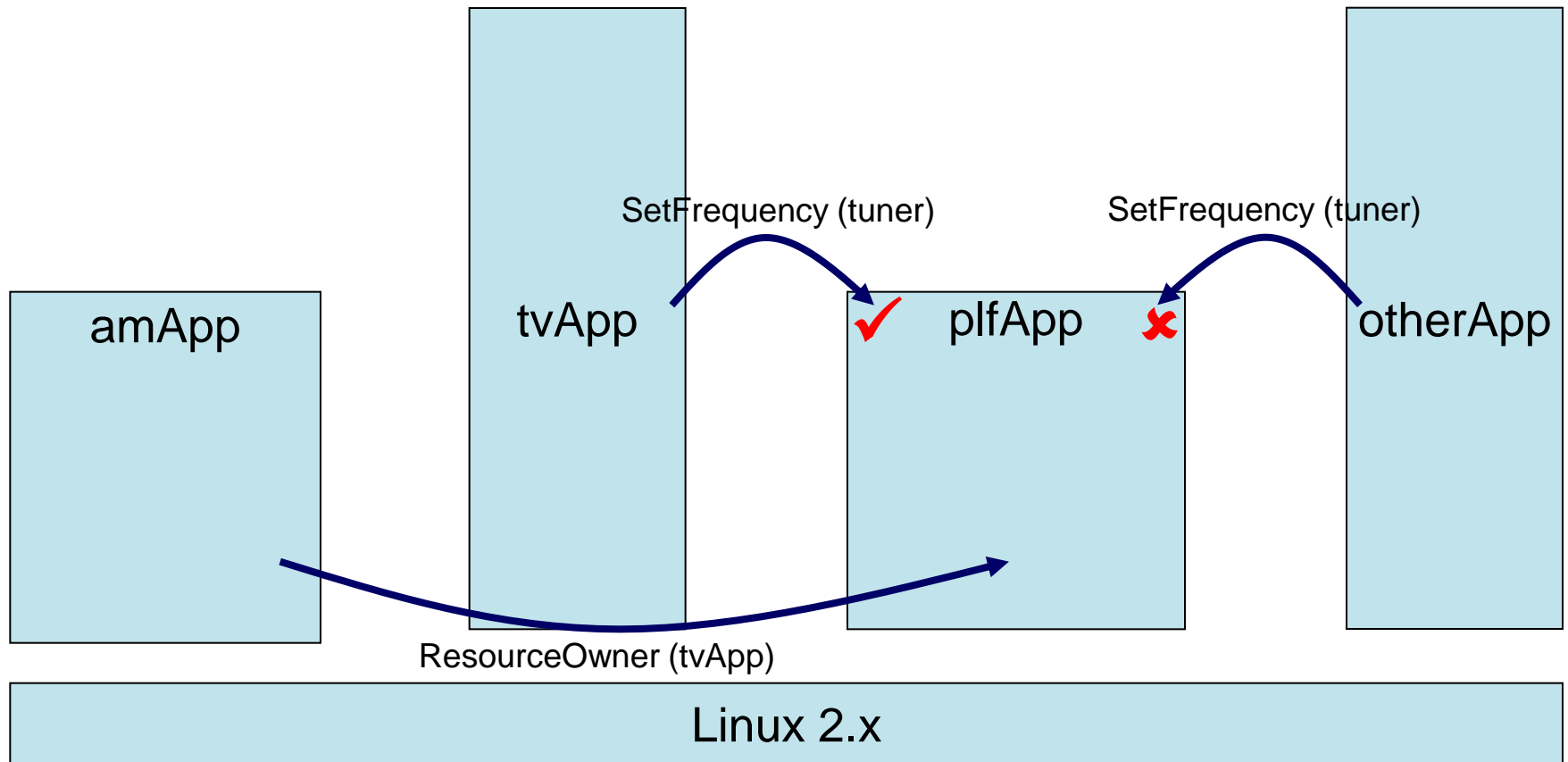
Managing resources

- There are implicitly managed resources by the kernel
 - TCP, flash, USB
 - Memory allocation
- There are explicitly managed resources
 - AV platform resources
 - Memory and CPU resources
- The explicitly managed resources
 - Have a statically defined execution behavior
 - Can by design limit the parallelism in the system
 - Require an explicitly resource controlled system

Dynamic Resource Management

- amApp has a fixed list of possible resource per application
 - Worst case requirements for an application (as per today)
- Application requests the resources it needs
 - Via an explicit list
- amApp assures resources if application is in focus
 - Only the resources that are allowed (fixed list of possible resources)
 - Can mean other applications loose resources (without stopping them)
- An application never releases resources
 - Always taken away by amApp

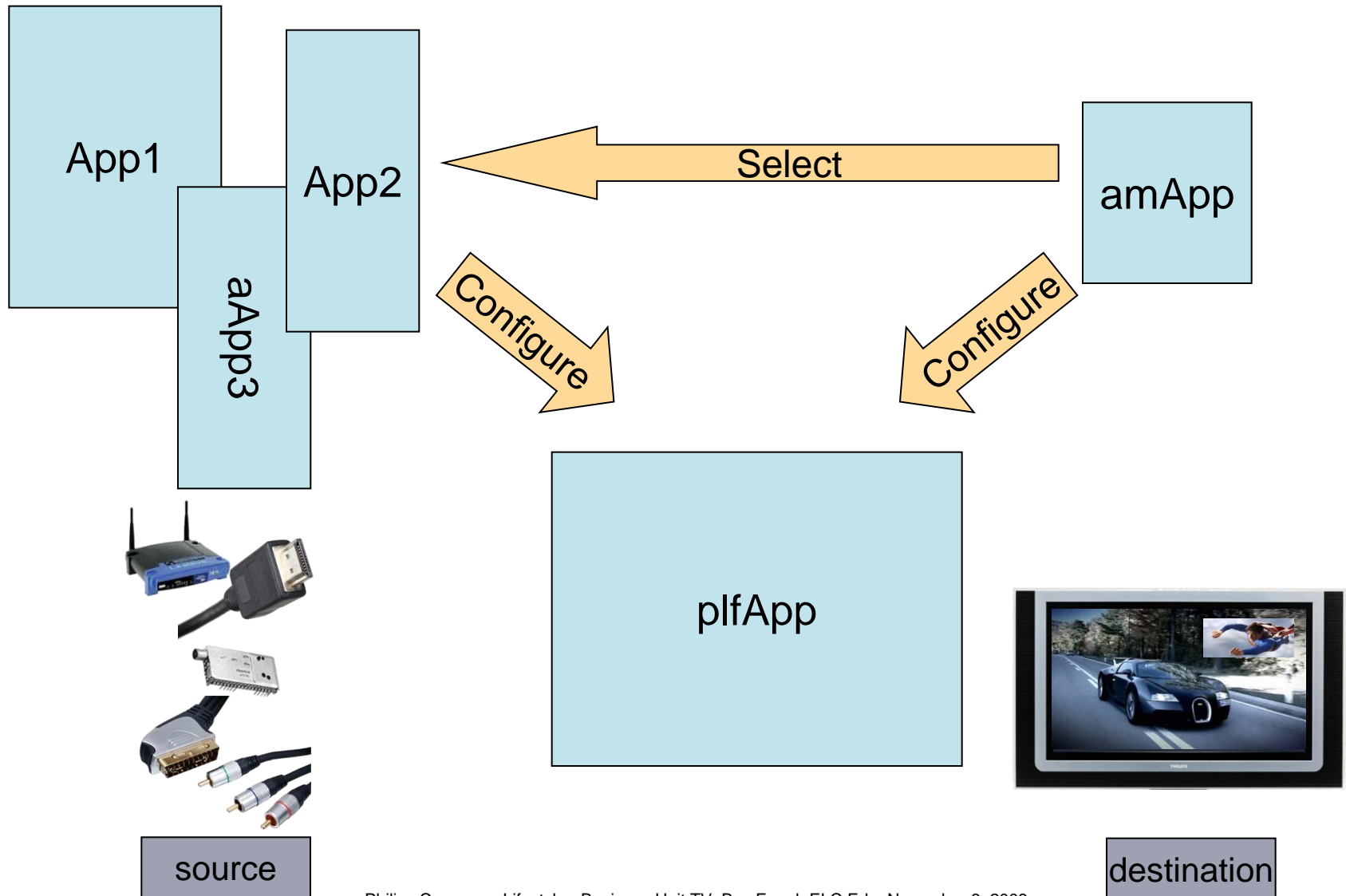
Multi Client Management



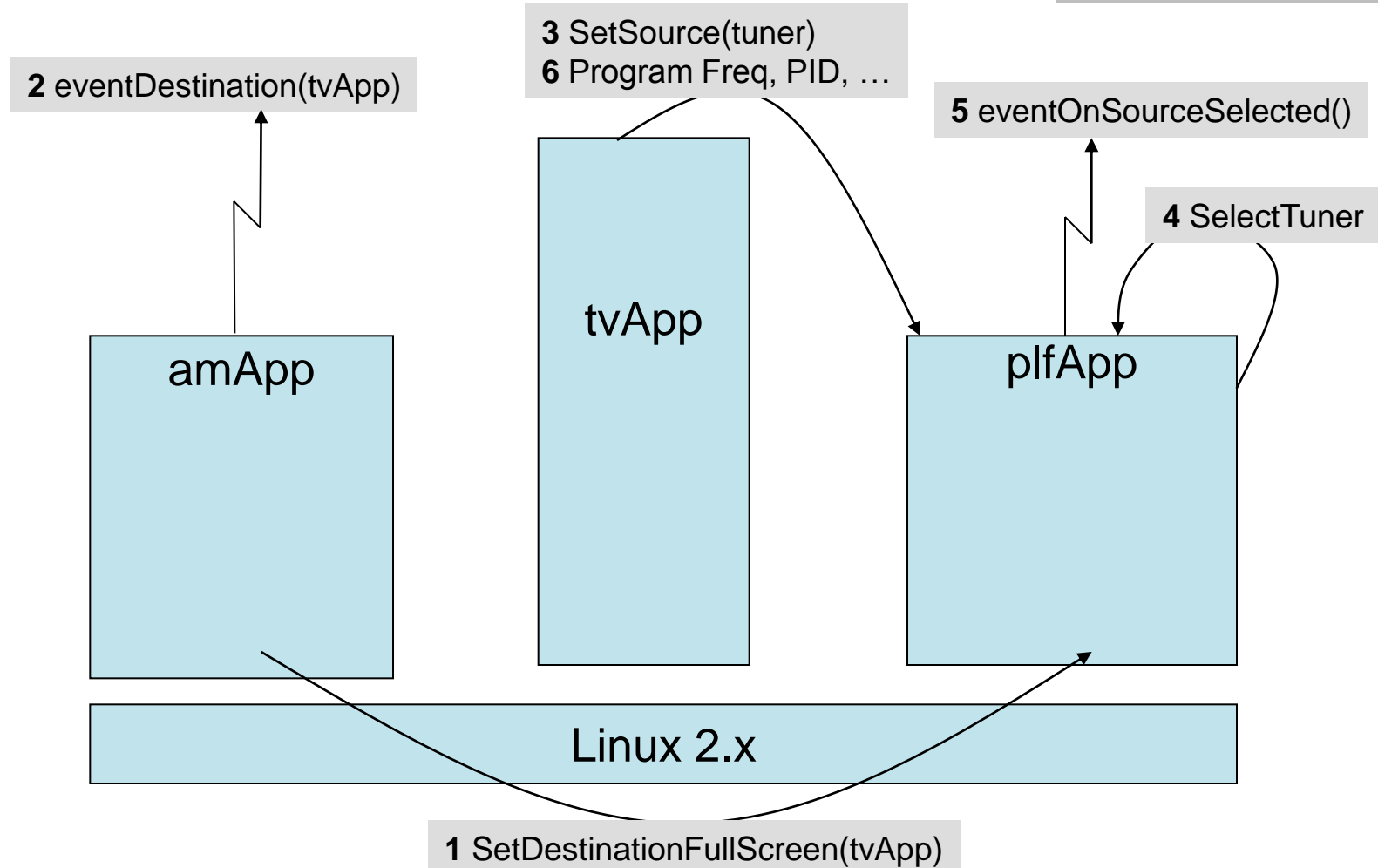
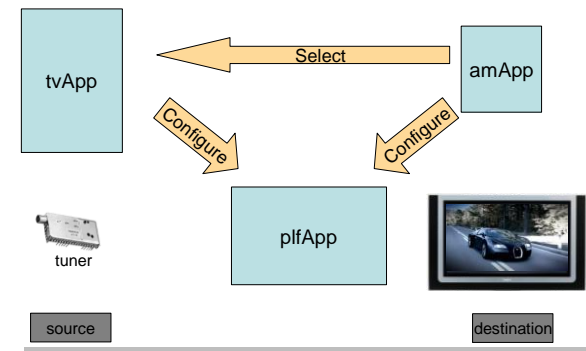
General Connection Management Concept

- Connection Management is split up in
 - Destination setup (full screen, window)
 - Source setup (HDMI, tuner, DLNA)
- Connection Management is distributed
 - Application Manager is responsible for destination setup
 - Application Manager is source unaware
 - Client applications are responsible for source setup
 - Client applications are destination unaware
- Identical approach for all use-cases
 - Audio, Video, and Graphics sources

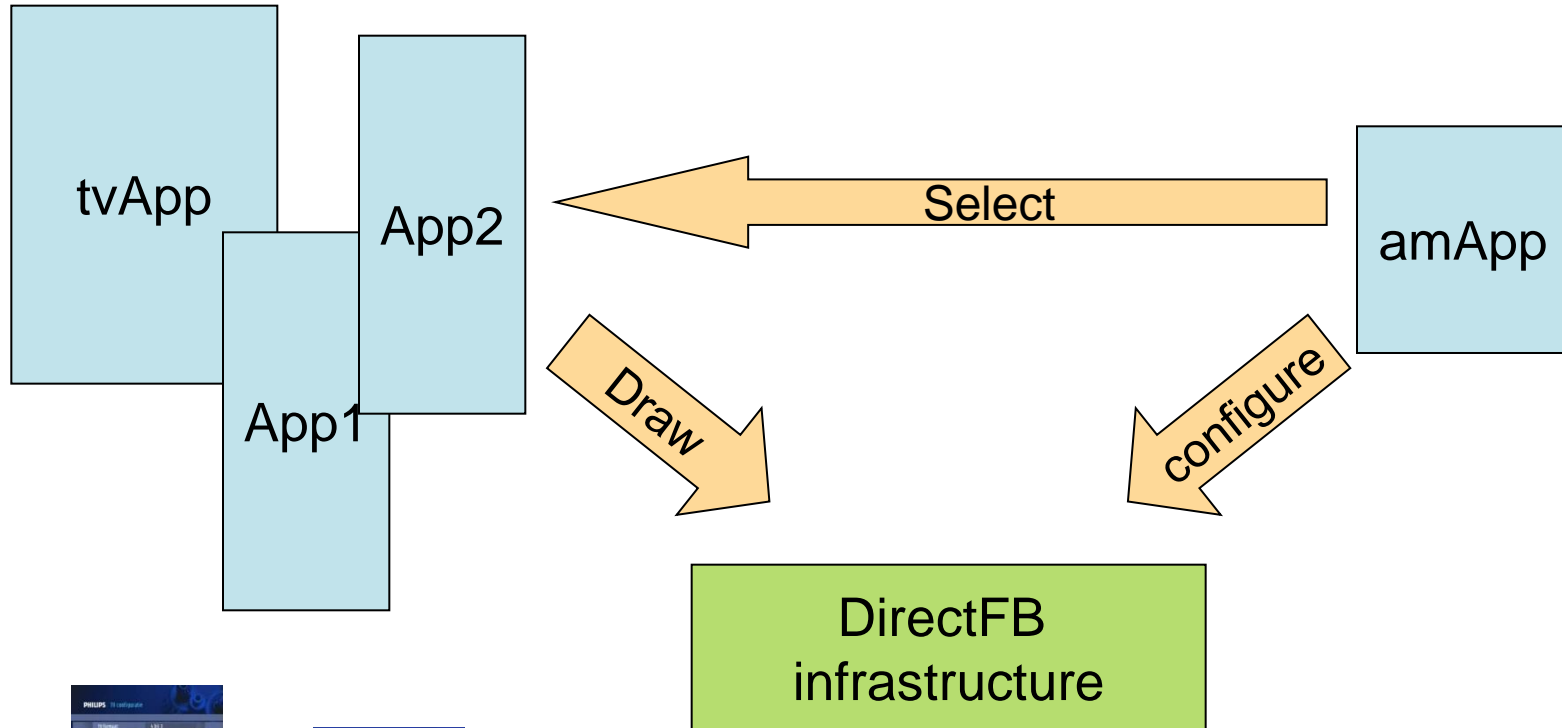
Audio/Video Connection management



Example



Graphics Connection Management

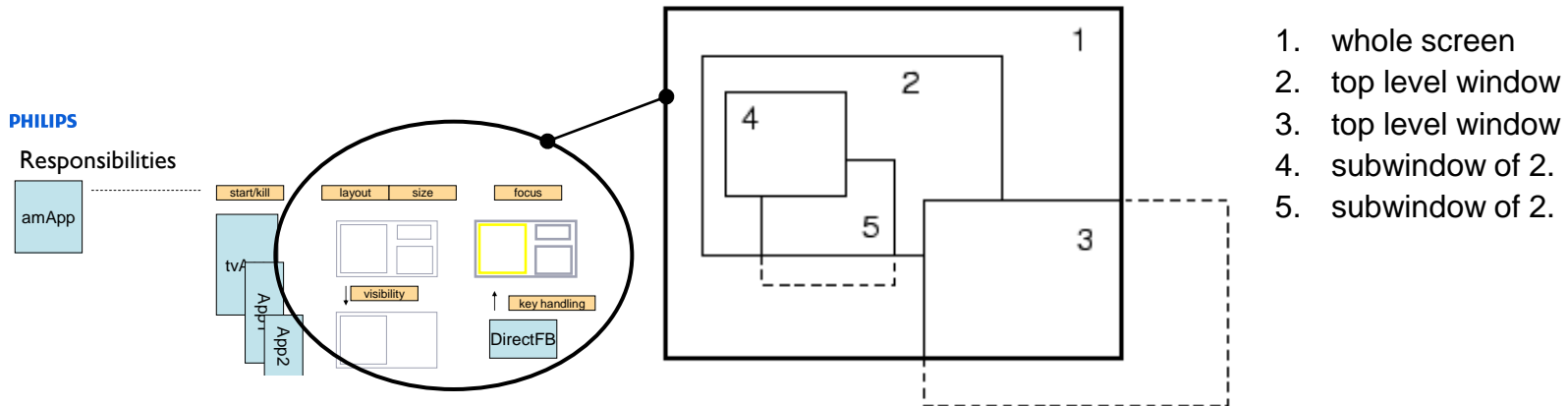


source



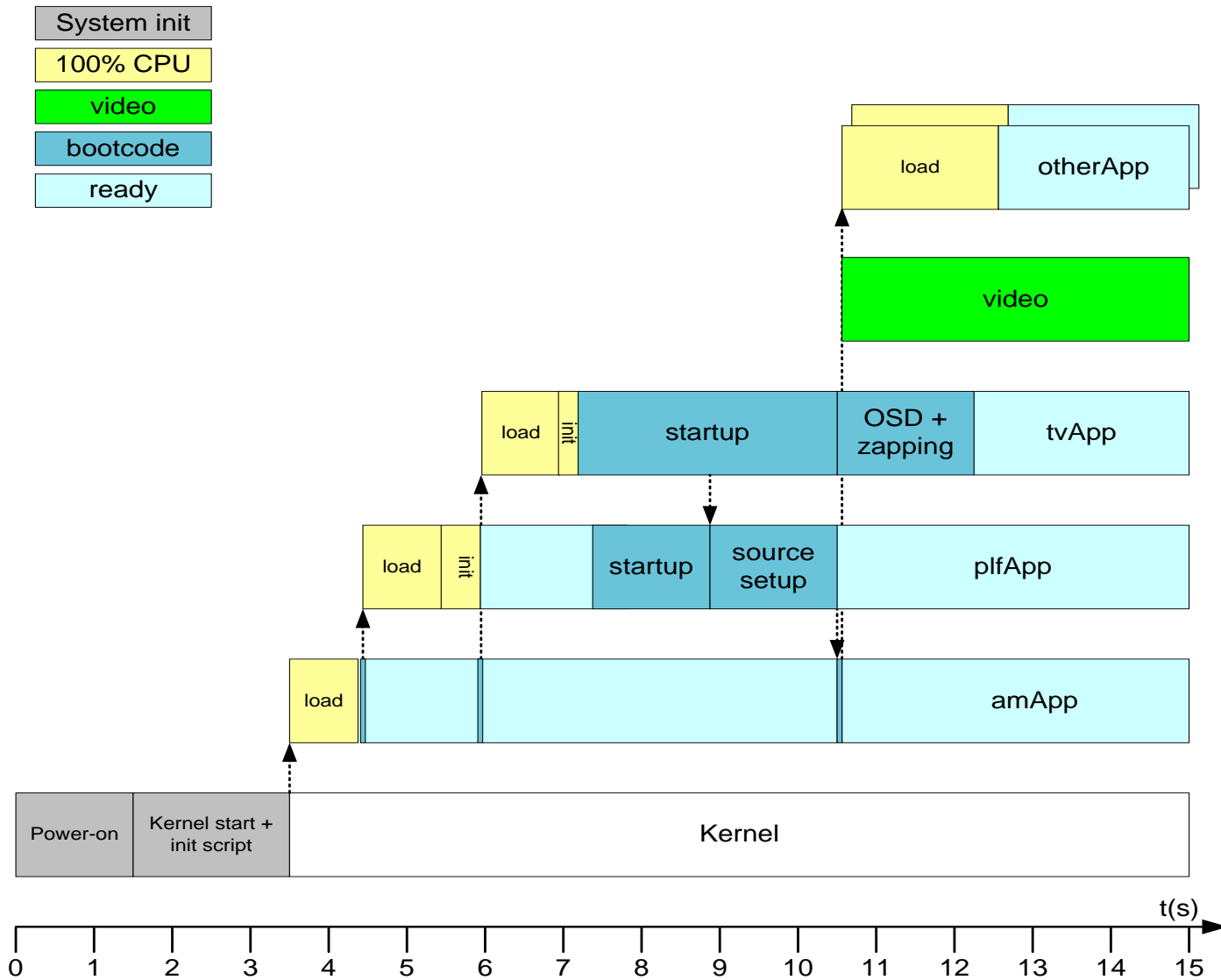
destination

Window Management



- Application manager responsible for all top-level windows
 - Z-order, focus, association, pixel format, layout
- Application are able to do their own internal window management
 - Applications use multiple windows for OSD like scenarios
 - Always slave of top-level window managed by application manager
- Application can create 1 or more sub-windows within 1 top-level window
 - A sub-window is linked to a top-level window

SPACE Boot Architecture



Key Optimizations Areas

- Use pre-linking for all SPACE applications
 - All SPACE applications are pre-linked (30% load time gain per application)
- File system optimizations
 - Optimize ECC and block loading
 - Separation of data and code partitions, reduce mount time
- Only initialize what is needed
 - Delayed mounting where possible of JFFS2, USB, Ethernet, etc
 - Use paging for all applications
- Parallelize tvApp and plfApp
 - Make sure during initialization the system runs at 100% CPU

Secure Boot

- Security/Robustness rules imply application integrity
 - Only start applications that have been verified to run on the system
- Use hash based algorithm (sha1) to verify application
 - Required for all executable code that could tamper with the system
 - Restricted to code that could access security data (key, license)
- We identified 2 options
 - Preferred solution: Accelerated RAM disk
 - Pre-load all applications, use HW to accelerate RAM disk verification
 - Verification and loading block based and fully parallel, no boot time increase
 - BUT, loading applications from RAM disk (instead of squashfs) to RAM: +2s
 - Traditional SPACE startup
 - Verification and loading block based, all done by Linux OS: +1s

Critical Success Factors For SPACE

- Avoid system requirements that require interweaved applications, like
 - Integrated zaplist for bolt-on and native zapping
 - Synchronized transition handling across applications
- Accept consistent behavior
 - We have to accept that not everything is possible in SW
 - No more exceptions to realize that one ‘important’ requirement
- Do not accept any synchronous communication between applications
 - A-synchronous events rigorously managed by architecture team
- Streamline organization towards SPACE
 - Specifically addressing reuse, responsibilities, release mechanism, diversity

The Transition Towards SPACE

- Major change in execution architecture
 - From single process (7 threads) to multi-process (unlimited threads)
 - From real time user space scheduling towards pre-emptive one
- Release an efficient multi-process resource management
 - Avoid explosion of memory requirements
 - From single client AV resources towards multi client
- Move from single application towards multiple small ones
 - Understanding where and how to split
 - From centralized control architecture towards distributed one
- Infrastructure change
 - From proprietary solution towards DirectFB
 - Enhancing DirectFB towards an embedded windowing system

Does SPACE Deliver It's Promise?

- Shorter lead time of new features
 - Orthogonality in functionality is essential part in TTM
 - Significantly reduced validation time of extension to basic system
 - Diversity simplification
 - Much easier to create differences between products
- Reduction of complexity
 - System is composed of logical self-contained building blocks
 - Less dependencies and far better to manage architecturally and project
 - No need to understand full system details for new features
- Improved portability
 - New HW platform impact restricted to plfApp
 - Execution architecture impact better contained locally

Next Steps

Dynamic application integration, system integrity

SPACE Rapid Prototyping Environment

- SPACE on PC enables fast prototyping of future TV use-cases
 - Enhanced multi-window
 - Compose multiple input streams
 - Real window rendering on PC
 - Vector Graphics and 3D graphics
 - OpenVG, OpenGL, SVG and Flash, DirectFB Water
 - All via PC cards and prototype extensions
- Fast turn-around of innovations
 - Space on PC is Space on target
 - Exception is performance
 - Graphics, key handling, plfApi are all identical
 - Communication, watchdogs, events are all emulated
 - Allows to develop and mature TV applications on PC

SPACE Run-time Configuration

- SPACE is designed such that applications run orthogonally
 - Applications do not influence each others dynamic behavior
- Via the existing infrastructure an application can easily be added
 - Application Manager does not restrict where applications are located
 - Dynamic resource manager enables plfAPI to be used by any one
 - DirectFB has a multi-window and multi-control concept
- There are some challenges still
 - Applications must be restricted in what they can access (sandboxing)
 - Applications must be validated before they can run

Ongoing Investigation

- Execute applications in a controlled environment
 - To restrict impact of application on system to ensure system stability
 - To prevent access to secure information (keys, DRM)
- Containment on various aspects
 - Limit files accessible
 - Control access to certain API's (plfApi, DirectFB, network, ...)
 - Constrain CPU usage and memory bandwidth usage
- We are investigating how to support sandboxing
 - Existing solutions seem to be an overkill for TV (eg virtualization, selinux)
 - Other solutions are not addressing the full scope (eg chroot)
 - Other initiatives are unclear if they bring an integral solution (eg cgroups)

Conclusion

Conclusion

- We described the need for a next generation software integration model
 - Addressing the emerging standards and services driven from STB domain
 - Enabling the ever growing content to be managed and accessed
 - Realizing the simplicity promise and easy control of functions
- SPACE addresses the integration challenge
 - Enabling orthogonal software integration
 - Realizing a simplistic resource model that is truly multi-client
 - Leveraging the power of a standard Linux infrastructure
- SPACE could be taken to the next level
 - Must understand how to get around security and stability restrictions

