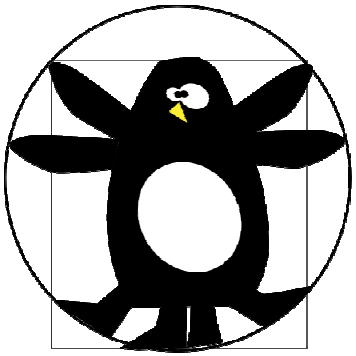
LINUXDAY 2013

abinsula

Fastboot in sistemi embedded



Sommario

abinsula

- Chi siamo
- Cos’è un sistema embedded
- Il tempo di boot
- Come funziona l’avvio del sistema
- Il bootloader
- Il kernel
- Il filesystem
- User space systemd

Chi siamo

abinsula

Abinsula è un'azienda che propone soluzioni per sistemi **Embedded**, nel campo della **Sicurezza Informatica** e sviluppo di applicazioni **Mobile** (Smartphone e Smart TV).

Nata nel marzo 2012 in poco più di un anno conta circa 20 unità fra dipendenti e collaboratori.

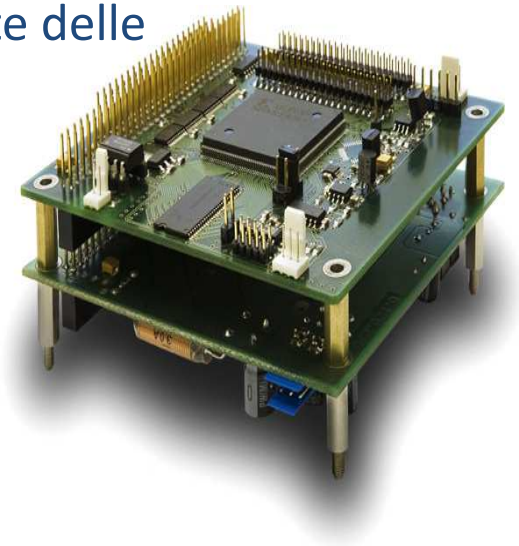
- Paolo Doz
  - Skill: sviluppo kernel device driver, customizzazioni distribuzioni Linux, Linux hacker
- Ilario Pittau
  - Skill: sviluppo device driver e hardware test suite, customizzazioni di sistemi linux embedded

Che cos'è un sistema embedded?

abinsula

Da Wikipedia…

"In elettronica e informatica, con il termine sistema embedded si identificano genericamente tutti quei sistemi elettronici di elaborazione a microprocessore progettati appositamente per una determinata applicazione spesso con una piattaforma hardware ad hoc, integrati nel sistema che controllano ed in grado di gestirne tutte o parte delle funzionalità richieste."

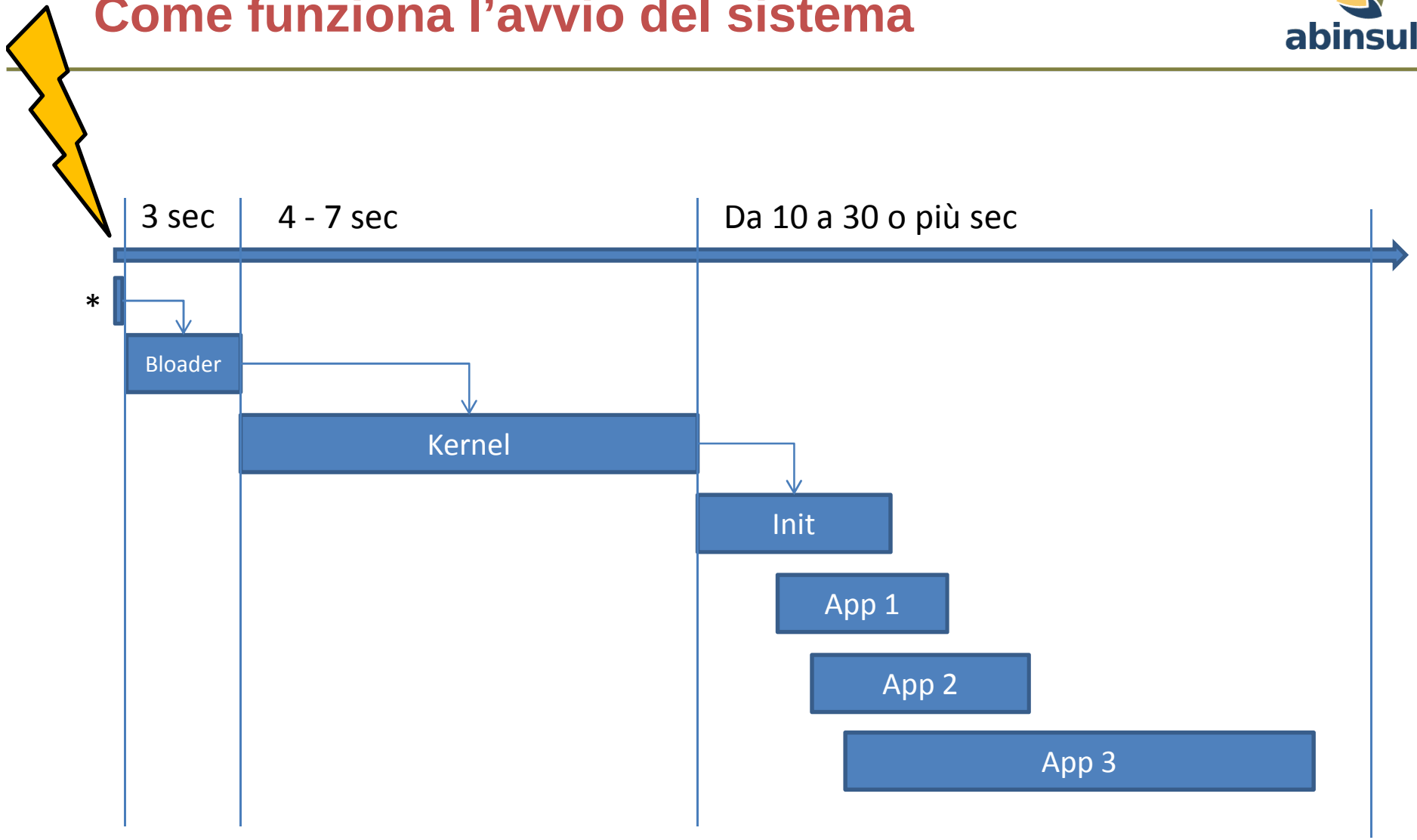


abinsula

Perché è importante il tempo di boot?

Senza tenere conto di quanto possa essere ben realizzato un dispositivo, il tempo necessario per passare dallo stato power off a quello di utilizzo è un punto critico che può inficiare la user-experience complessiva.

Come funziona l'avvio del sistema



* Firmware – 10ms

Il bootloader

abinsula

Da Wikipedia…

Il boot loader è quel programma che, nella fase di avvio (boot) del computer, carica il kernel del sistema operativo dalla memoria secondaria alla memoria primaria, permettendone l'esecuzione da parte del processore e il conseguente avvio del sistema

- Bootloader per sistemi embedded e non:
  - Das U-boot (PPC, ARM, x86, MIPS, AVR32, Blackfin, Motorola 68000 e altre architetture). E' il più usato nel mondo embedded
  - BareBox (PPC, ARM, x86, MIPS, Blackfin). Evoluzione di U-boot, struttura molto simile al kernel Linux
  - GRUB (x86, PowerPc). Utilizzato di default dalla gran parte delle distribuzioni Linux

Il caricamento del bootloader

abinsula

- Può essere caricato in RAM ed eseguito oppure eseguito direttamente dallo storage (XIP)
- Può essere ospitato su differenti tipi di memorie:
  - NOR
    - Utilizzate nel caso si voglia eseguire del codice direttamente dalla memoria. Di solito molto piccole (< 8MB). Veloci in lettura random, limitati cicli di scrittura, costose.
  - NAND
    - Scrittura a blocchi gestione via SW. Economiche e più compatte delle NOR
  - eMMC/SD
    - Stesse caratteristiche delle NAND ma controllo blocchi via HW. Facile aggiornare il filesystem di un dispositivo embedded perché rimovibili

Esecuzione del bootloader

abinsula

- Il bootloader ha il compito di inizializzare il sistema
  - Clock
  - RAM e MMU
  - Chip presenti sulla board (es controller audio/video, tuner, hub usb)
- Mette a disposizione una serie di strumenti molto utili in fase di sviluppo. Ad esempio avere la possibilità di utilizzare un filesystem remoto (NFS) o utilizzare una penna usb per il trasferimento dati.

Ottimizzazioni

abinsula

- Rimozione console e autodelay all'avvio
- Rimozione inizializzazioni dispositivi non usati durante la fase di boot (I2C, FLASH, USB, EXT3, DOS, FAT, SPI, SATA, VIDEO)
- Rimozione di tutti i comandi non utilizzati
- Dimensione iniziale uboot -> 330KB
- Dopo la rimozione delle funzionalità non utilizzate -> 93KB (riduzione del 70%)
- Tempo di boot ridotto da 300 ms a 25 ms

Ottimizzare ma non troppo!

abinsula

Non tutte le inizializzazioni possono essere rimosse, per alcuni componenti il kernel si aspetta che siano già inizializzati.

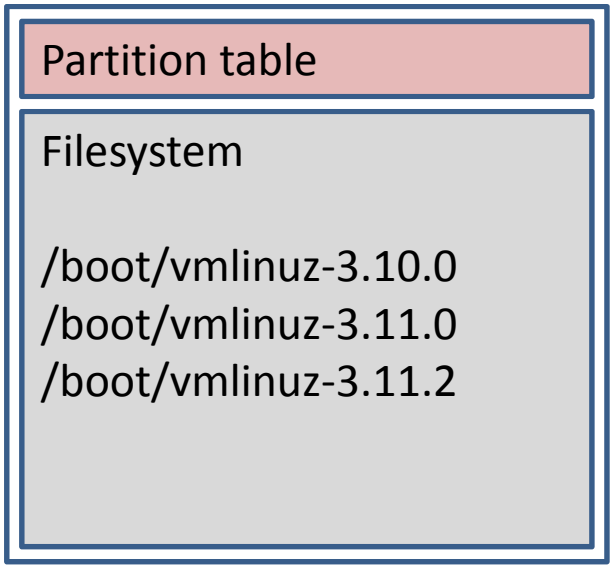
Es: ETH del processore Freescale iMX6. Se il chip ethernet non viene inizializzato in fase di boot, il kernel non riesce a configurare correttamente il networking

```
[ 55.004031] ------------[ cut here ]------------
[ 55.009740] WARNING: at net/sched/sch_generic.c:254
dev_watchdog+0x298/0x2b8()
[ 55.018721] NETDEV WATCHDOG: eth0 (fec): transmit queue 0 timed out
```

Dove è memorizzato il kernel?

abinsula

Architettura tradizionale

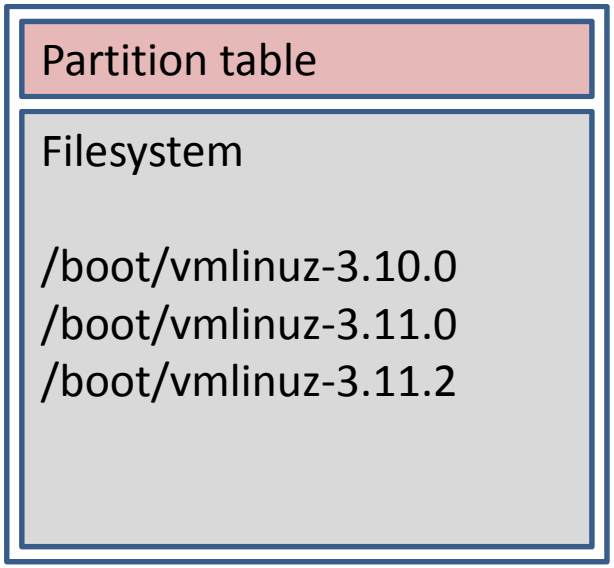


Il bootloader legge la partition table, monta il filesystem, carica l'immagine del kernel e poi la esegue

Dove è memorizzato il kernel?

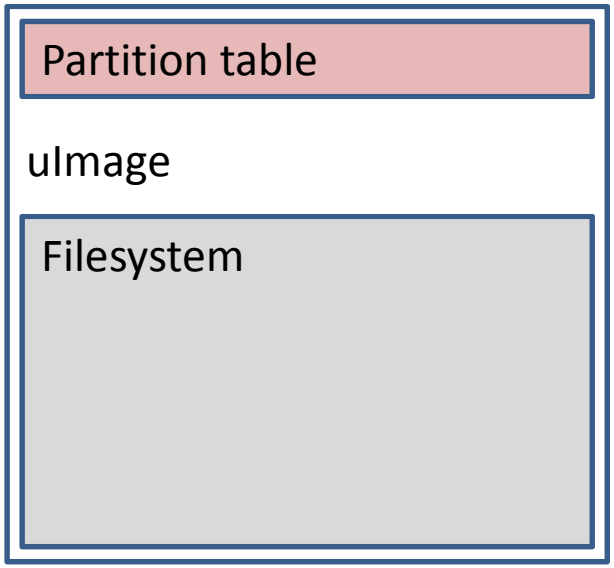
abinsula

Architettura tradizionale



Il bootloader legge la partition table, monta il filesystem, carica l'immagine del kernel e poi la esegue

Architettura ottimizzata



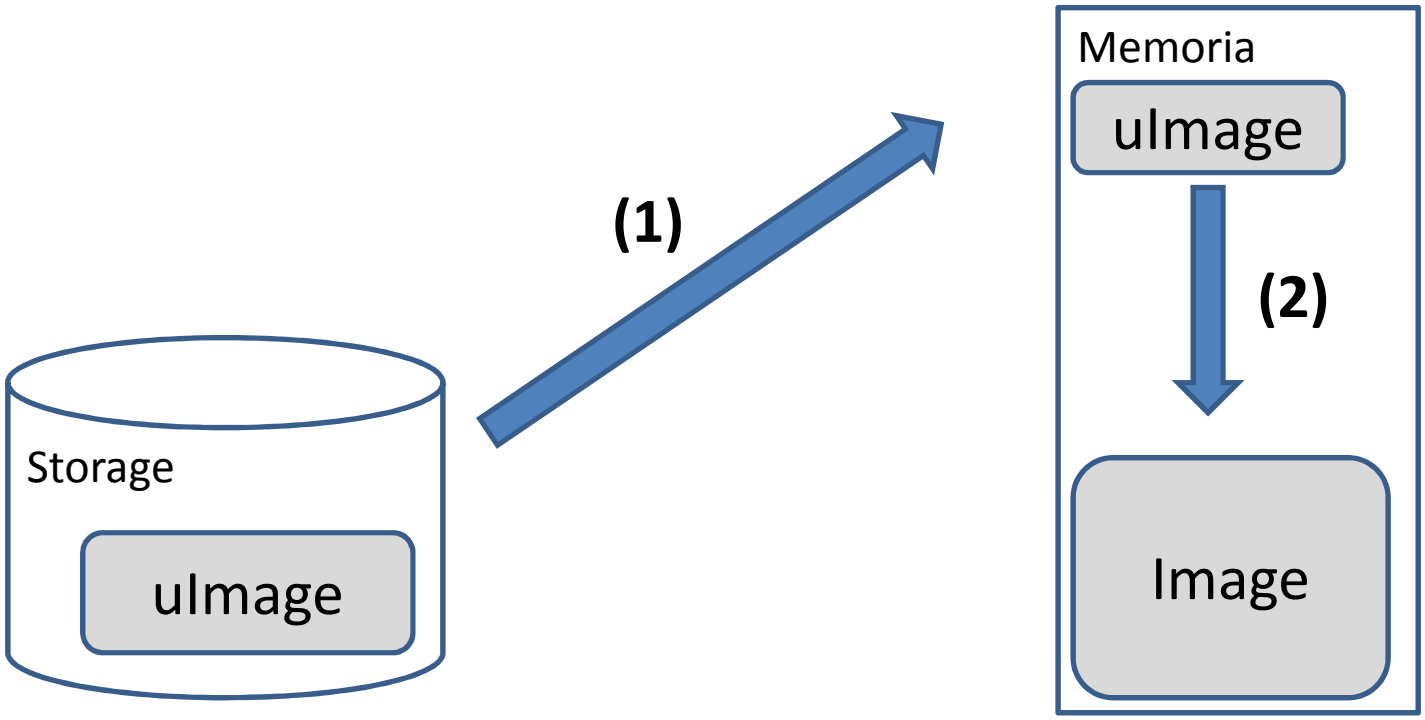
Il bootloader legge dati raw direttamente dalla memoria risparmiando il discovery delle partizioni e il mount del filesystem

```
bootcmd=mmc read ${loadaddr}
0x800 0x1000 ; bootm
${loadaddr} ;
```

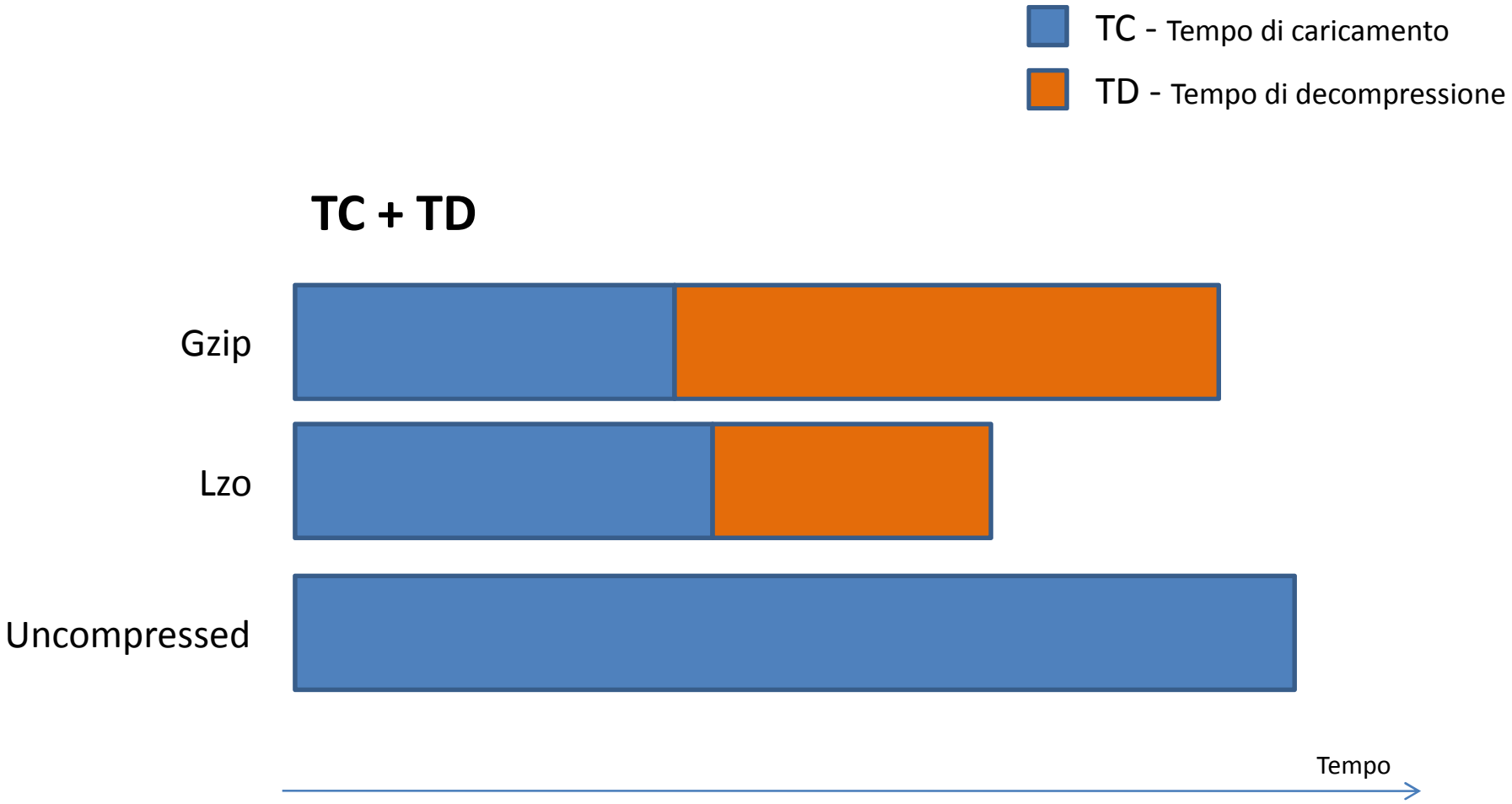
Caricamento del kernel

abinsula

- Viene letto dal dispositivo di storage, caricato in RAM (1), decompresso (2) ed infine eseguito
- Tipi di compressione: LZMA, GZIP, BZIP2, XZ, LZO



Kernel compresso o no?



Esecuzione

abinsula

- Lettura della cmdline
- Inizializzazione dell'hw
  - Reset e configurazione dei componenti
- Mount del rootfs
  - Initrd vs initramfs vs filesystem standard
- Esecuzione del processo init per proseguire il boot

Ottimizzazione 1/2

abinsula

- Caricamento e decompressione tramite DMA
- Eliminazione delle virtual console superflue (100ms)
- Skip della calibrazione “loops per jiffy” – lpj – (700ms)
- Finito il tuning del sistema rimuovere le stampe di debug con "quiet" (200ms)

Ottimizzazione 1/2

abinsula

- Caricamento e decompressione tramite DMA
- Eliminazione delle virtual console superflue (100ms)
- Skip della calibrazione “loops per jiffy” – lpj – (700ms)
- Finito il tuning del sistema rimuovere le stampe di debug con "quiet" (200ms)

Esempio di cmdline

```
console=ttymxc1,115200 vmalloc=40M
consoleblank=0 rootwait root=/dev/mmcblk0p1
lpj=7905280 quiet
```

Ottimizzazione 2/2

abinsula

Riduzione della dimensione del kernel:

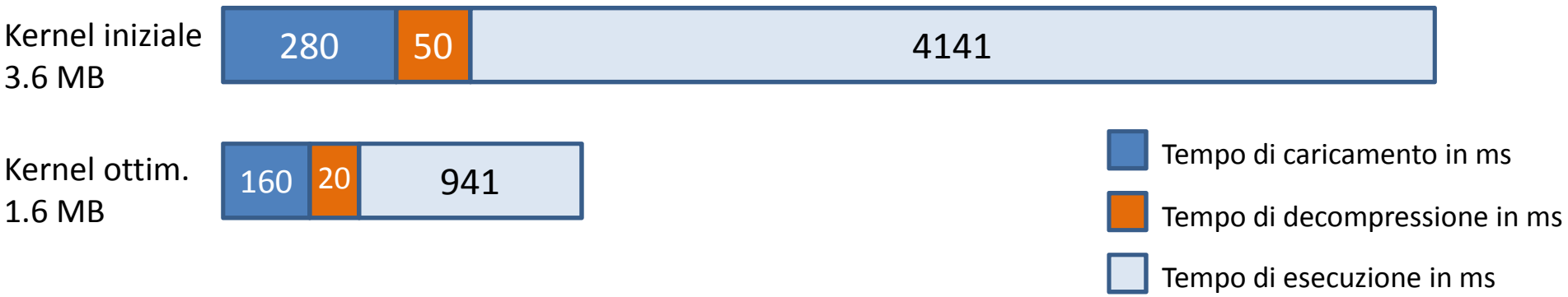
- Rimozione dei driver inutilizzati o chip/bus non presenti: bluetooth, wi-fi, pci, flash, scsi, sata, i2c;
- Rimozione del supporto di feature inutilizzate: sysVipc, initrd, freq-scaler, netfilter, filesystems, debugfs, printk;
- Dove possibile usare driver modulari: ethernet, CAN

Ottimizzazione 2/2

abinsula

Riduzione della dimensione del kernel:

- Rimozione dei driver inutilizzati o chip/bus non presenti: bluetooth, wi-fi, pci, flash, scsi, sata, i2c;
- Rimozione del supporto di feature inutilizzate: sysVipc, initrd, freq-scaler, netfilter, filesystems, debugfs, printk;
- Dove possibile usare driver modulari: ethernet, CAN



Kernel e filesystem

abinsula

L'immagine del kernel (uImage) può essere composta da:

- **Solo kernel**
  - Pro: file molto piccolo, veloce da caricare
  - Contro: il mount del filesystem finale rallenta un po' il boot
- **Kernel + initrd**
  - Pro: è molto veloce montare il filesystem in RAM
  - Contro: richiede un driver per la lettura del filesystem (es ext2), per questo è stato rimpiazzato da initramfs
- **Kernel + initramfs**
  - Pro: usa tmpfs, riconosciuto automaticamente dal kernel, molto utilizzato nei sistemi embedded.
  - Contro: complicato effettuare lo switch root alla fine della fase di boot per montare il filesystem finale

Filesystem 1/2

abinsula

Come districarsi nella giungla dei filesystem embedded?
Esistono tanti filesystem adatti per sistemi embedded, ognuno con caratteristiche diverse per ogni utilizzo:

- Ext2, Ext3
  - Adatti ad ogni storage, molto robusti e supportati.

- ReiserFS, Ext4
  - Non particolarmente veloci a causa del journaling, inadatti per sistemi embedded.

- Jffs2, UbiFS, SquashFS, F2FS
  - Adatti a memorie flash e/o partizioni read-only

Filesystem 2/2

abinsula

Differenza di mount su MMC con diversi Ext filesystem:

ext2
Startup finished in 991ms (kernel) + 2.592s (userspace) = 3.584s

ext3
Startup finished in 941ms (kernel) + 2.553s (userspace) = 3.494s

ext4
Startup finished in 1.125s (kernel) + 2.639s (userspace) = 3.765s

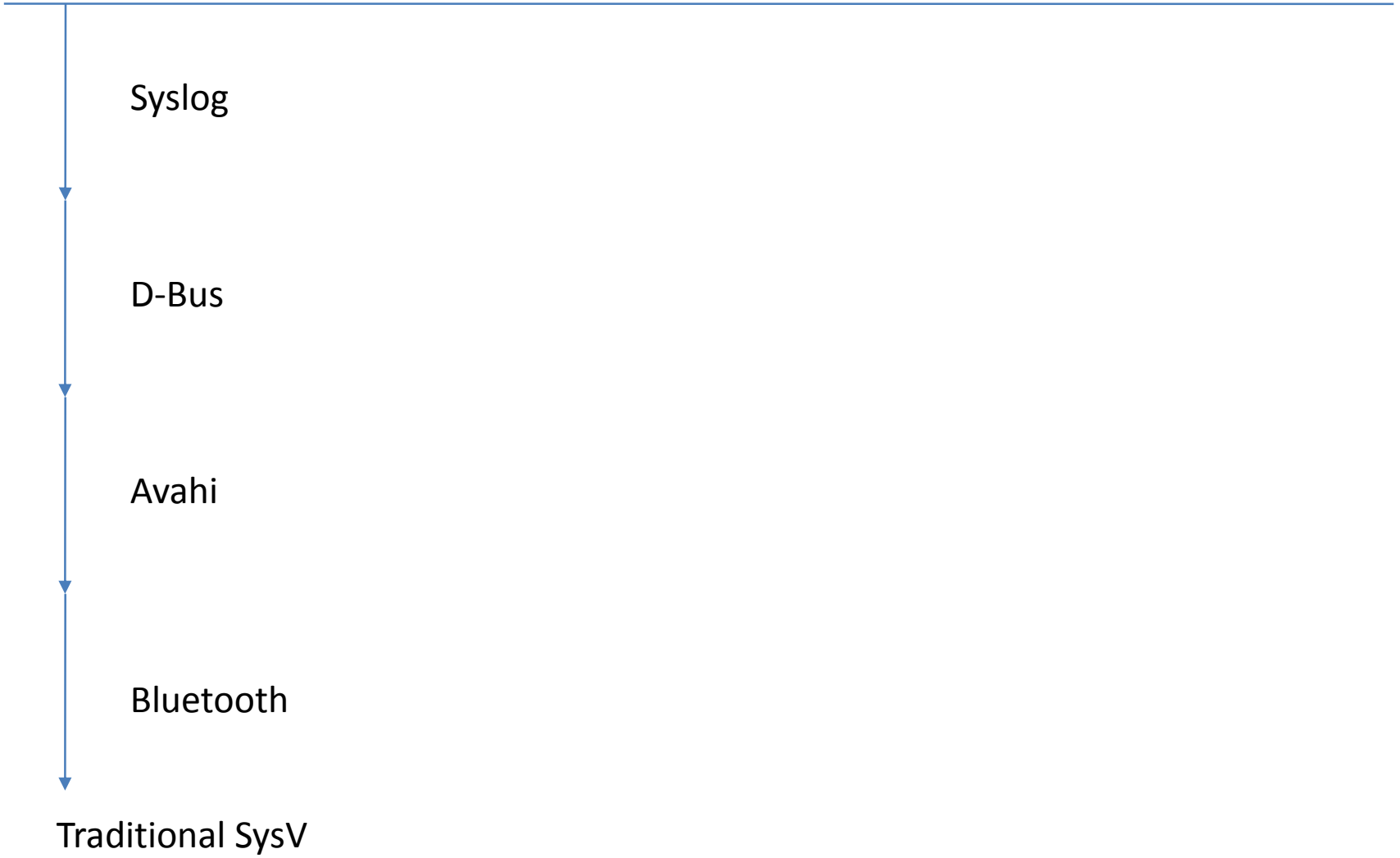
systemd 1/2

abinsula

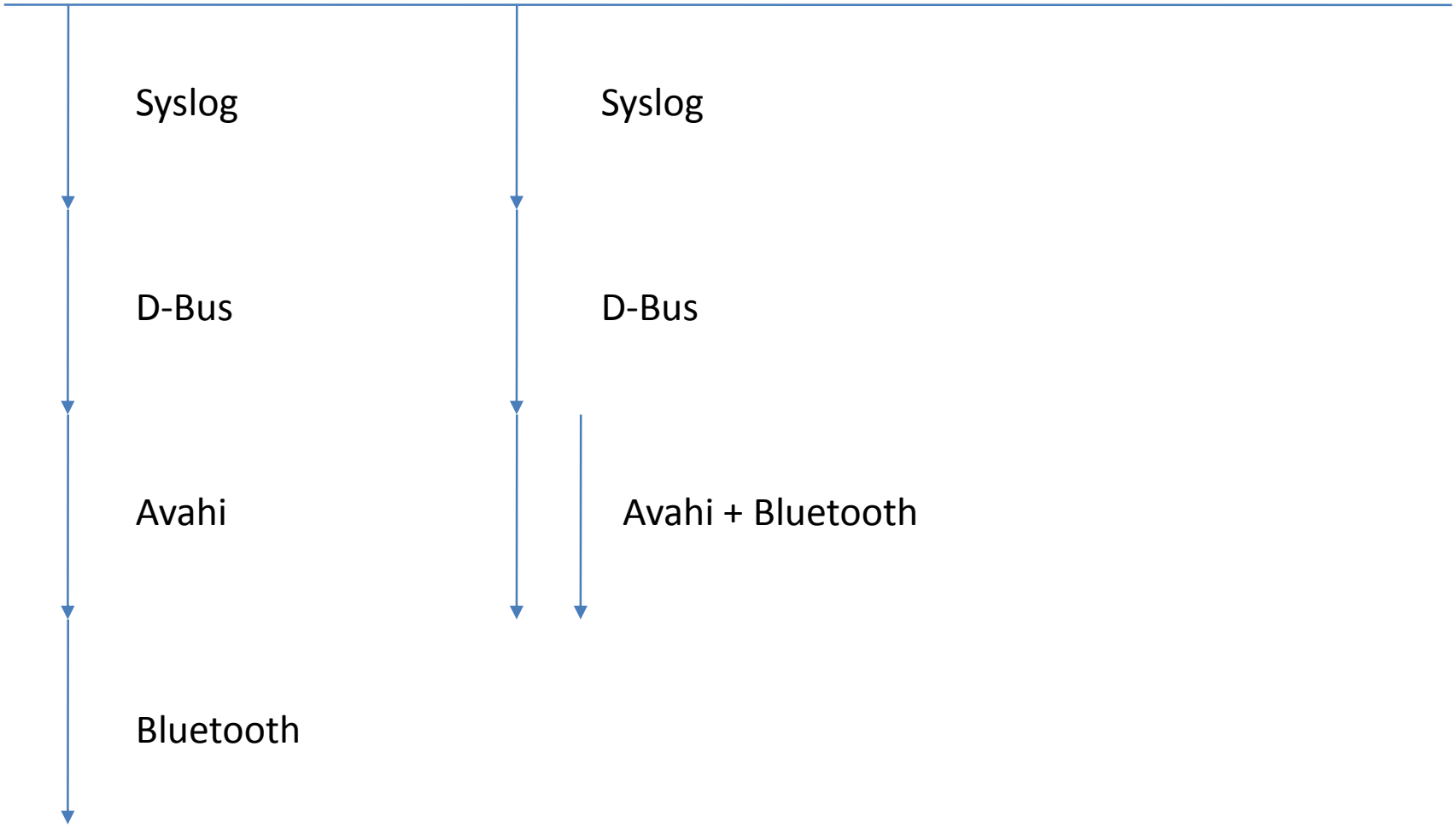
Dal sito di systemd :

systemd è un sistema e gestore dei servizi per Linux, compatibile con gli init scripts SysV e LSB. systemd permette una parallellizzazione aggressiva dei task, usa socket e D-Bus per far partire i servizi, permette di far partire i demoni on-demand, tiene traccia dei processi usando i Linux cgroups, supporta lo snapshotting e il restoring dello stato del sistema, mantiene i mount e automount point e implementa una elaborata logica di controllo dei servizi. Mira a diventare un sostituto per sysVinit.

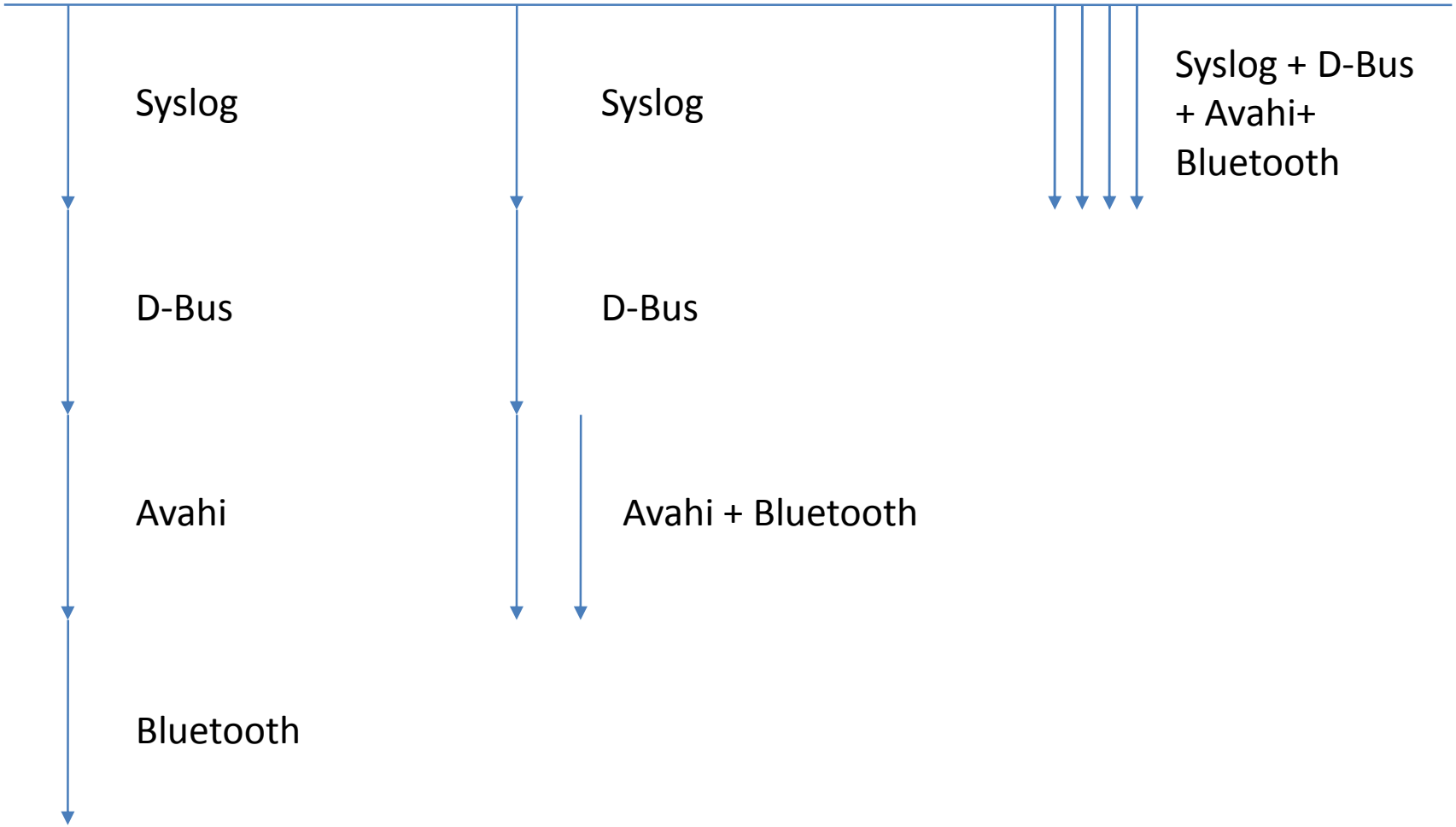
Confronto gestori di init



Confronto gestori di init



Confronto gestori di init



systemd 2/2

abinsula

Il funzionamento base

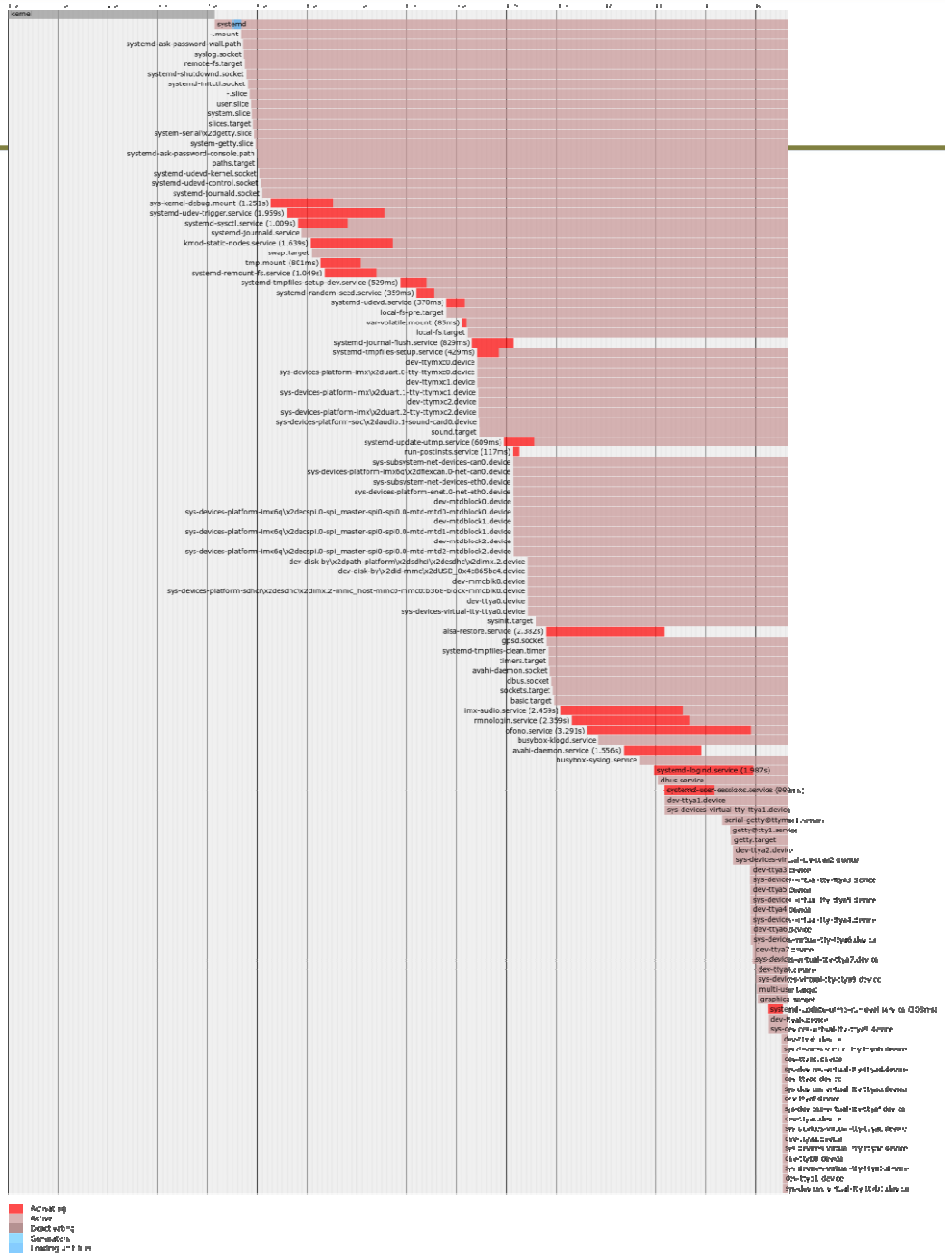
Il processo systemd avviato come primo processo legge la cmdline per capire il target da raggiungere e accede a "/lib/systemd/system/" per leggere i file di configurazione:
- systemd.target: sono punti statici da raggiungere durante il boot.
- systemd.[auto]mount: rappresentano i dispositivi che devono essere montati, possono essere montati anche su richiesta.
- systemd.service: rappresentano i servizi, normalmente chiamano un demone o un generico applicativo.
- systemd.socket: implementano i socket su cui systemd bufferizza o ascolta le richieste per avviare i .service.

Service di systemd

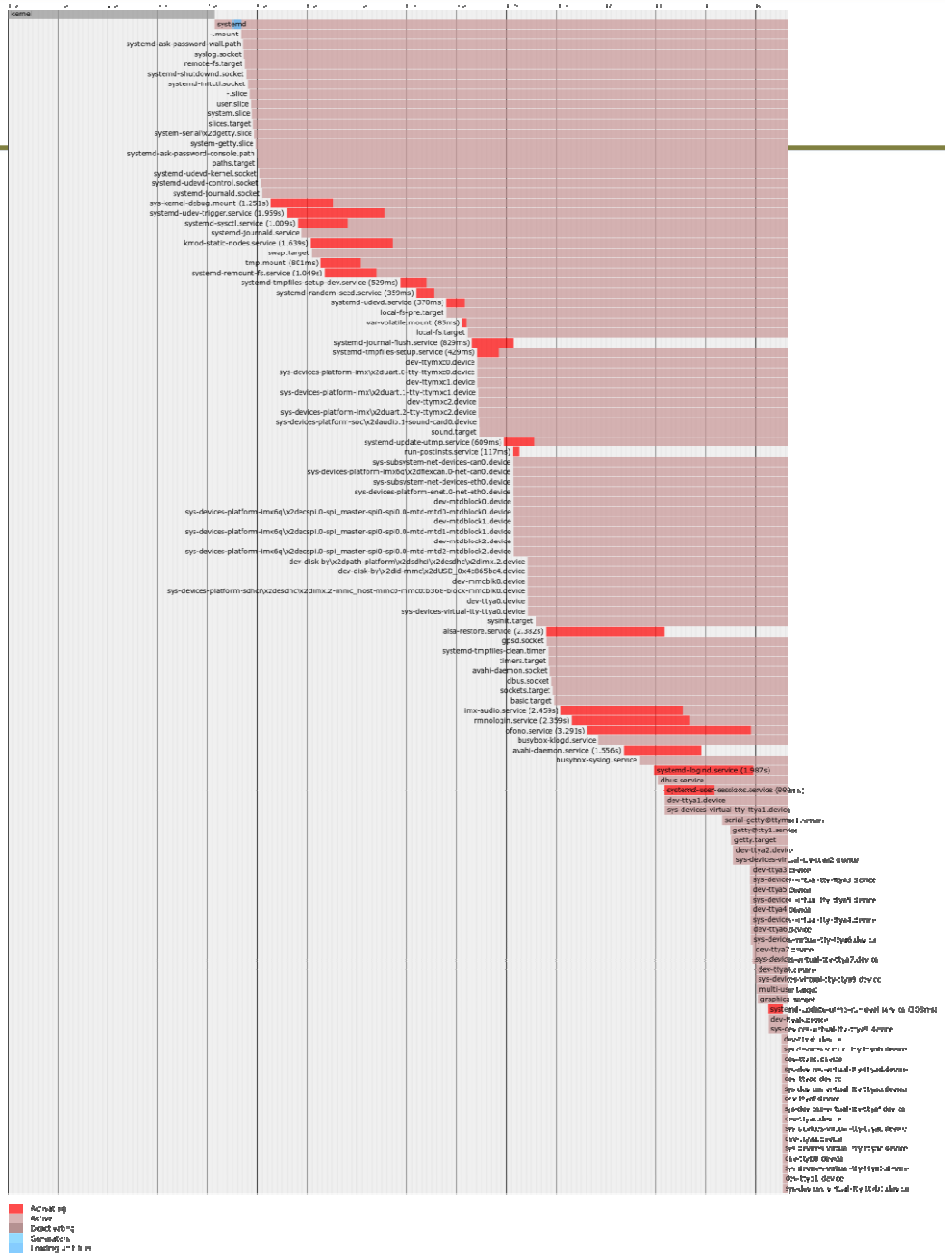
abinsula

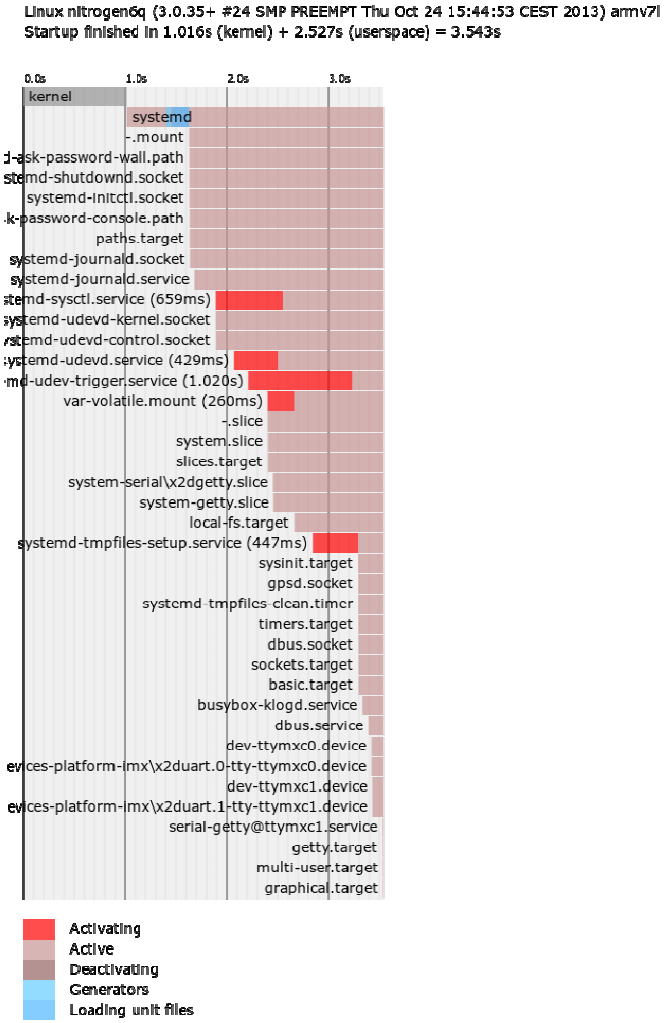
Esempio del service relativo a syslog

```
[Unit]
Description=System Logging Service
Wants=busybox-klogd.service

[Service]
EnvironmentFile=-/etc/default/busybox-syslog
ExecStart=/sbin/syslogd -n $OPTIONS
Sockets=syslog.socket

[Install]
WantedBy=multi-user.target
Also=busybox-klogd.service
Alias=syslog.service
```



abinsula




Linux nitrogen6q (3.0.35+ #24 SMP PREEMPT Thu Oct 24 15:44:53 CEST 2013) armv7l
Startup finished in 1.016s (kernel) + 2.527s (userspace) = 3.543s
0.0s
1.0s
2.0s
3.0s
kernel
systemd
-.mount
systemd-shutdownd.socket
systemd-initctl.socket
paths.target
systemd-journald.socket
systemd-journald.service
systemd-sysctl.service (659ms)
systemd-udevd-kernel.socket
systemd-udevd-control.socket
systemd-udevd.service (429ms)
var-volatile.mount (260ms)
-.slice
system.slice
slices.target
system-serial\x2dgetty.slice
system-getty.slice
local-fs.target
systemd-tmpfiles-setup.service (447ms)
sysinit.target
gpsd.socket
systemd-tmpfiles-clean.timer
timers.target
dbus.socket
sockets.target
basic.target
busybox-klogd.service
dbus.service
dev-ttymxc0.device
dev-ttymxc1.device
serial-getty@ttymxc1.service
getty.target
multi-user.target
graphical.target
Activating
Active
Deactivating
Generators
Loading unit files

Ottimizzazioni 1/2

abinsula

Come fare il debug dello startup?

```
# systemd-analyze plot
# systemd-analyze critical-chain
# systemctl [start/stop/status] <nomeservice>
# journalctl
```

Quali servizi ci servono?

- Rimozione di NFC(1s), Ofono (1s), Avahi(800ms), alsa-restore(200ms), video (700ms).
- Rimangono getty, d-bus, applicazione audio

Ottimizzazioni 2/2

abinsula

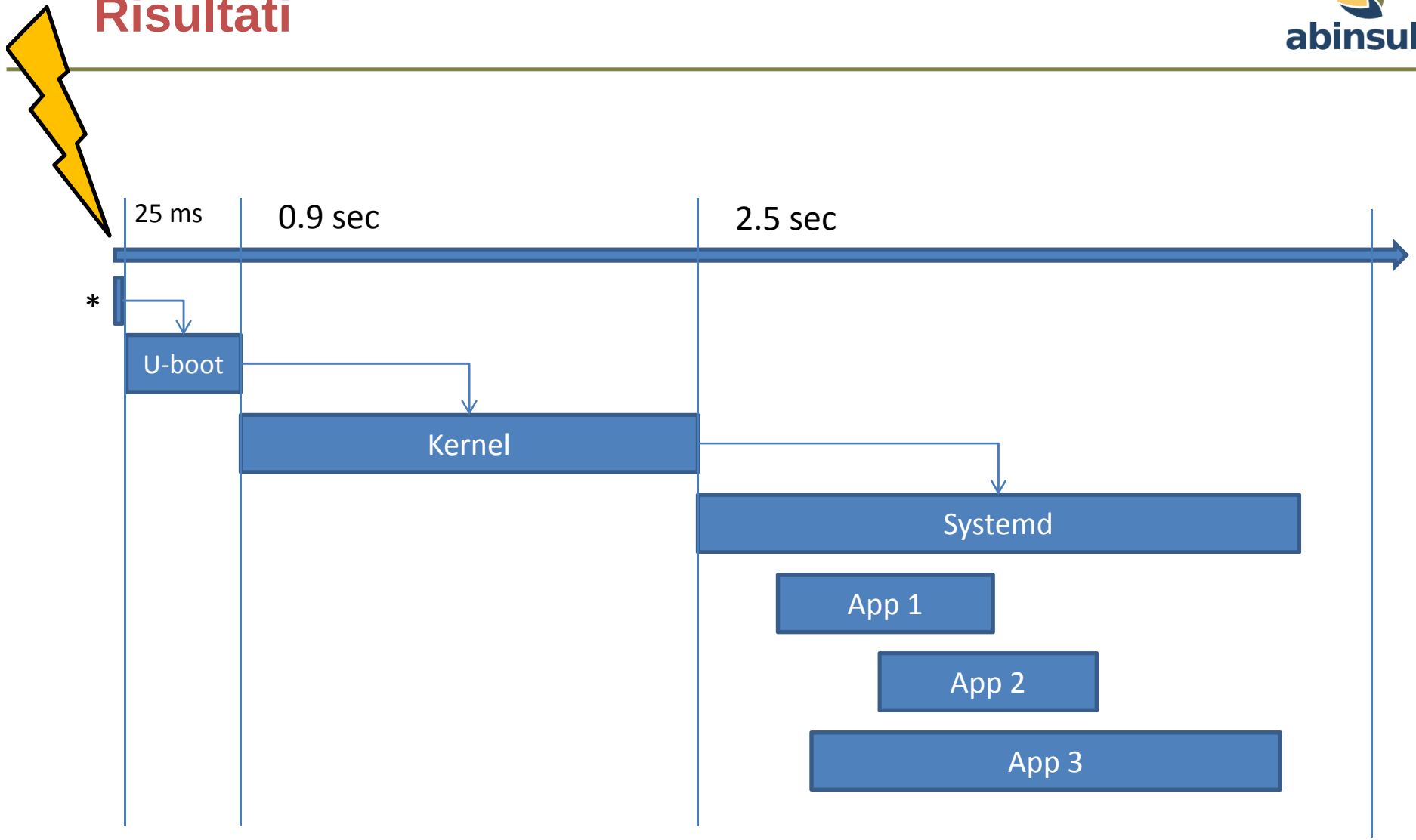
Che configurazioni statiche si possono fare?

- Creazione statica delle cartelle/file/dev temporanei. (529ms)
- Linkare le cartelle temporanee il più possibile per montare pochi tmpfs (600ms)
- Rimozione dei postinits (supporto ai vecchi script SysV) (170ms)
- Mount con fstab, Random-seed (1s)
- Modalità single-user o rimozione dei blocchi al login (3s)

Cosa si deve montare?

- proc e sysfs: viene fatto automaticamente da systemd
- tmpfs: /var/volatile /run
- devpts, debugfs: possono essere disabilitati (1.2s)

Risultati



* Firmware – 10ms

Conclusioni

abinsula

- Si può migliorare ancora?

Conclusioni

abinsula

- Si può migliorare ancora?
  - Ovviamente si
    - Creazione dei device e rimozione di udev
    - Modifiche nel codice sorgente
    - Sistema di init basato su script custom

Conclusioni

abinsula

- Si può migliorare ancora?
  - Ovviamente si
    - Creazione dei device e rimozione di udev
    - Modifiche nel codice sorgente
    - Sistema di init basato su script custom
- Conviene?

Conclusioni

abinsula

- Si può migliorare ancora?
  - Ovviamente si
    - Creazione dei device e rimozione di udev
    - Modifiche nel codice sorgente
    - Sistema di init basato su script custom
- Conviene?
  - Dipende
    - Tempi
    - Costi

Demo

abinsula

- Solo teoria?
- Board Freescale iMx 6, SD e fs creato con Ability

Ability

abinsula

Ability è una meta-distribuzione creata da Abinsula. E' basata su Open Embedded ed è fortemente orientata al mondo dei dispositivi embedded.

Tempi di boot ridotti

Efficienza del power management

Supporto device mobile

Affidabilità

GENIVI compliant

Grazie….

http://www.abinsula.com

paolo.doz@abinsula.com ilario.pittau@abinsula.com