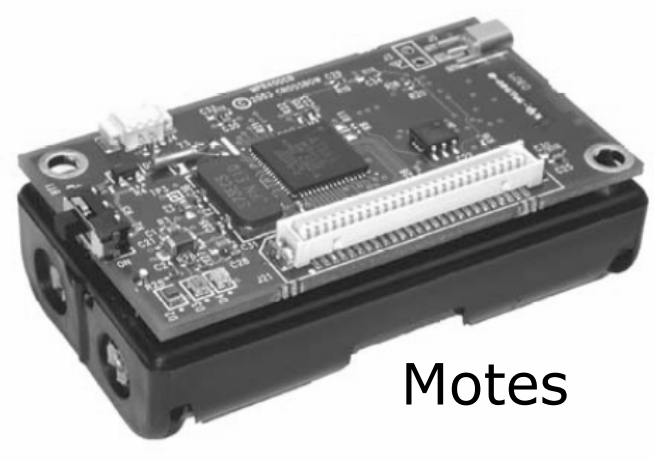
Ubiquitous Resources Abstraction using a File System Interface on Sensor Nodes

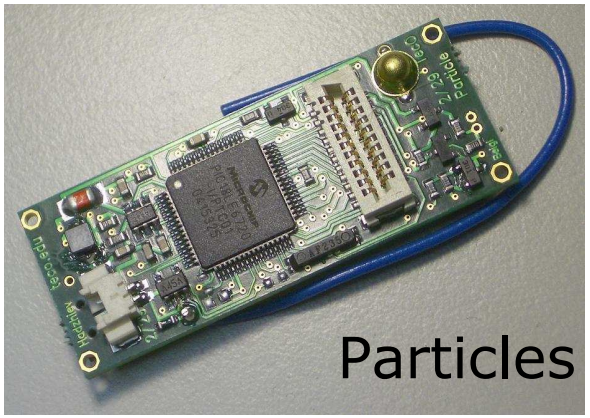
Till Riedel, Christian Decker
TecO, University of Karlsruhe
Institut for Telematics
Telecooperation Office (TecO)
www.teco.edu

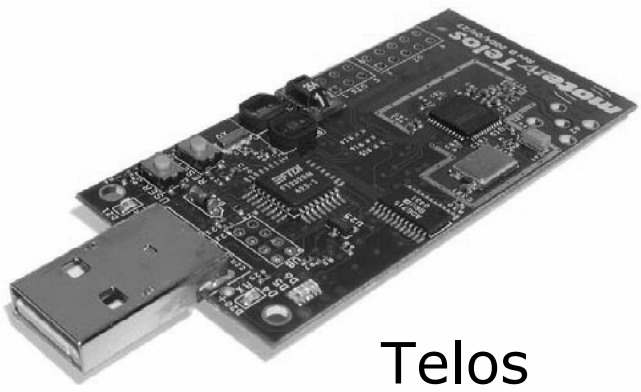
Outline

- **Analysis**
- **Design of a Resource Abstraction**
- **Evaluation**
- **Applications**

Embedded Sensor Devices in Ubicomp


Motes

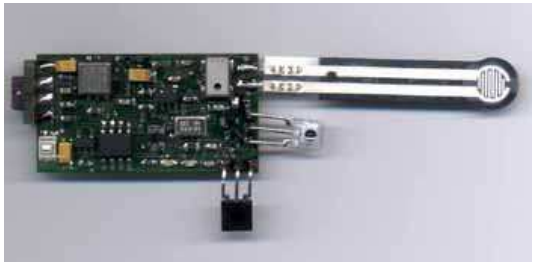

Particles


Telos

- Limited computing power, 8-bit microcontroller
- Few kilobytes … 512 kilobytes of Flash memory
- Customized radio protocols
- Battery powered
- Extensible by various sensors

Analysis

- **Big variety of sensors**



- **Growing APIs**

- **Experiences with developers**
  - Hesitate to develop code for „new“ sensors
  - Stick to the stuff they know
  - Use example program

Ubicomp Development – State-of-the-Art

- Lightweight OS (e.g. TinyOS)
  - OS shields resources, e.g. sensors
  - No direct access
  - Communication through events -> event dispatching
- Library based access models
  - Abstract access functions
  - Direct access, virtually no overhead
- But… shielding/abstraction causes still confusion

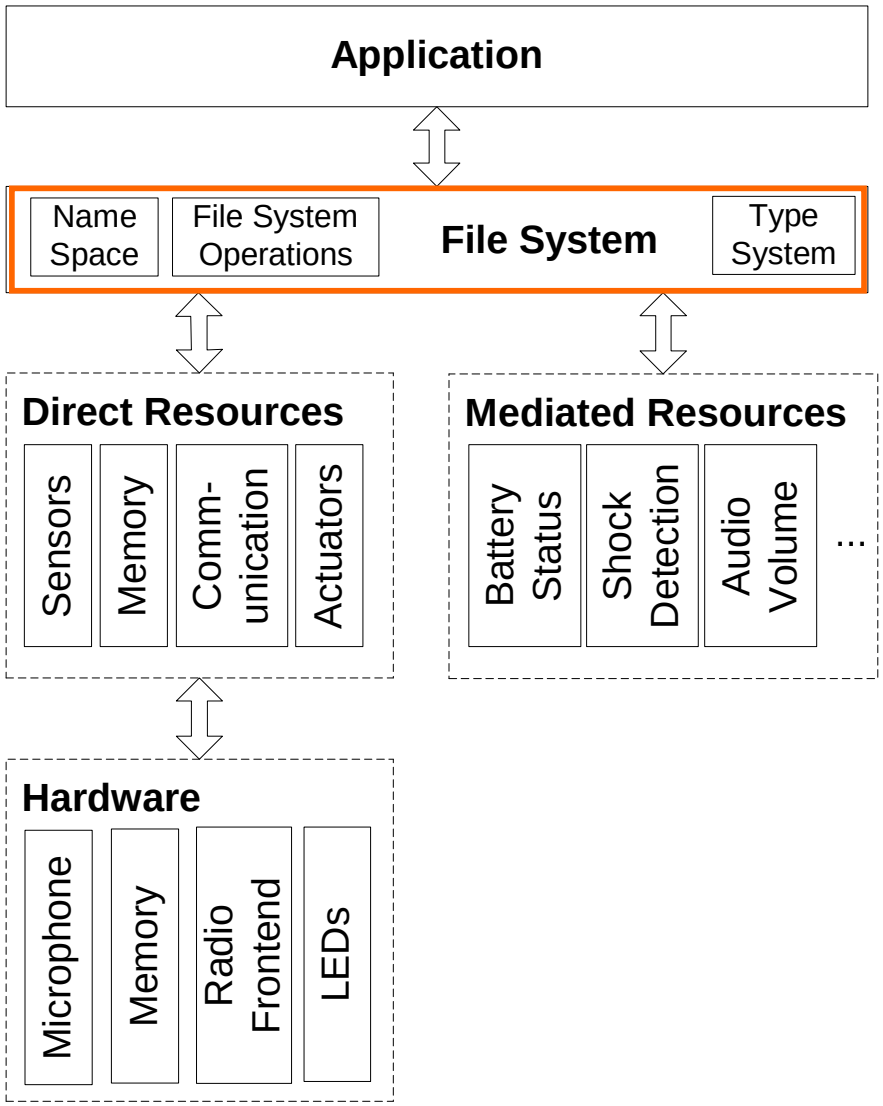
File System for Sensor Nodes

Design principles

- Developer in the center
- Support developer to follow the simplest way
- Easy-to-understand interfaces -> generic

Support in two ways

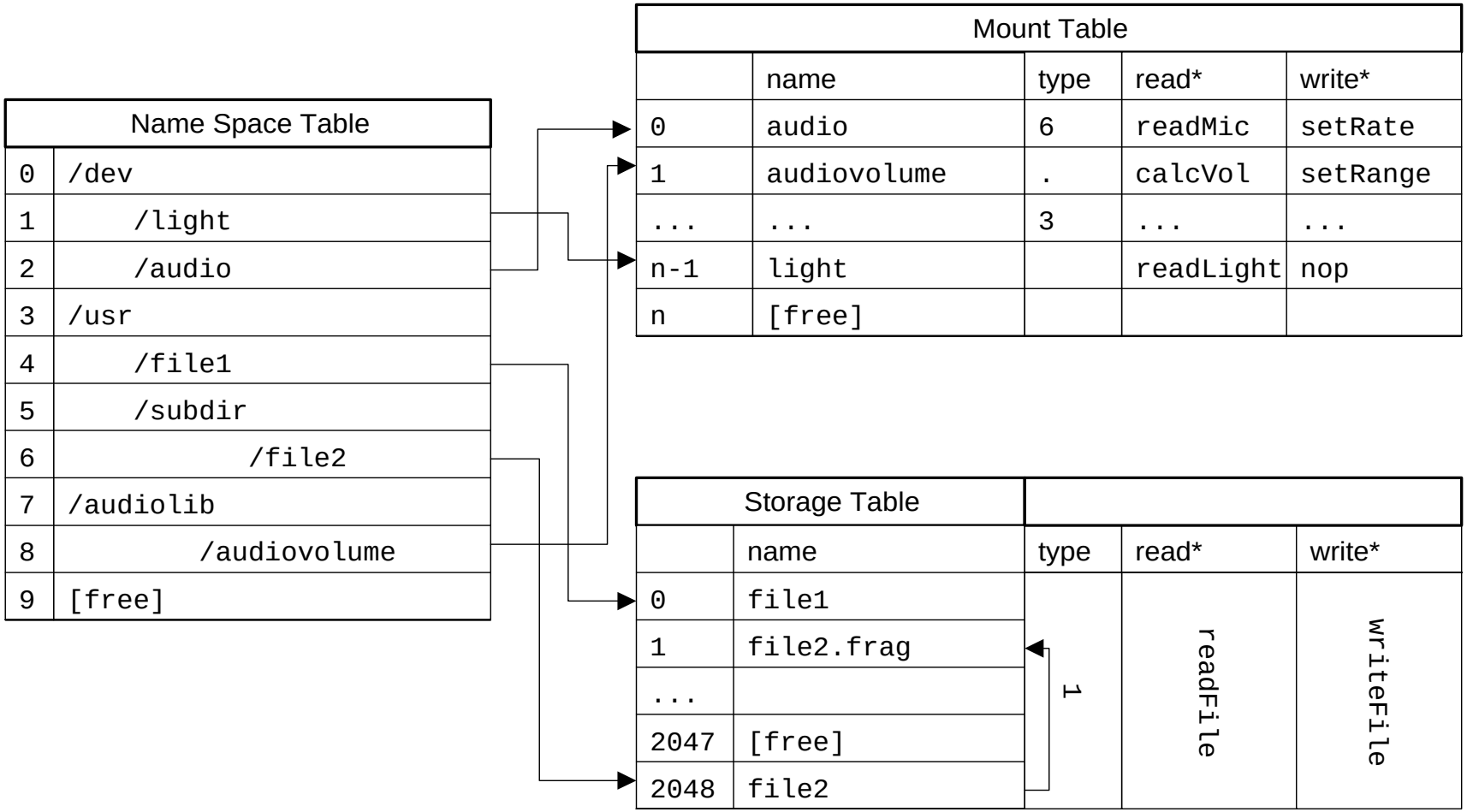
- Uniform representation of all resources
- Uniform access model



File System Interface

Operation	Explanation
size_t read (int fd, void* buf, size_t n)	Reads n data bytes from the resource identified by fd to buf; returns number of bytes or -1 if error occurred
size_t write (int fd, void* buf, size_t n)	writes n data bytes from buf to the resource fd; returns number of bytes or -1 if an error occurred
int open(char* resource_path)	Returns a descriptor for the resource; -1 if it is not valid.
int close(int fd)	Frees the descriptor of a resource; -1 if fd is not valid
int getType(int fd)	Returns the type of a resource fd; -1 if fd is not valid.
int mount (char* resource_path, int type, (*pFunc) read, (*pFunc) write)	Creates a resource in the name space. Type and function pointers to their specific read and write operations are given. -1 is returned if the resource_path already exists.
int umount(char* resource_path)	Removes a resource; -1 is returned if it is not valid.

Namespace and Ressources



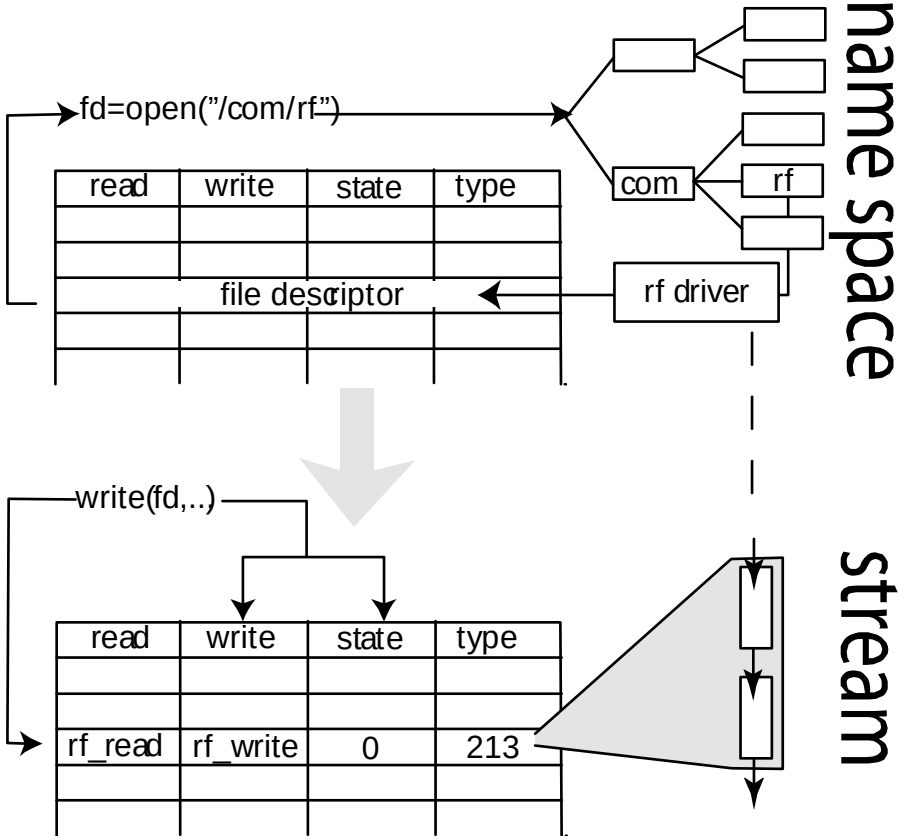
File System – Access Model

Name space

- Hierarchical
- Textual representation of resources
- Combine freely resource and specific behavior

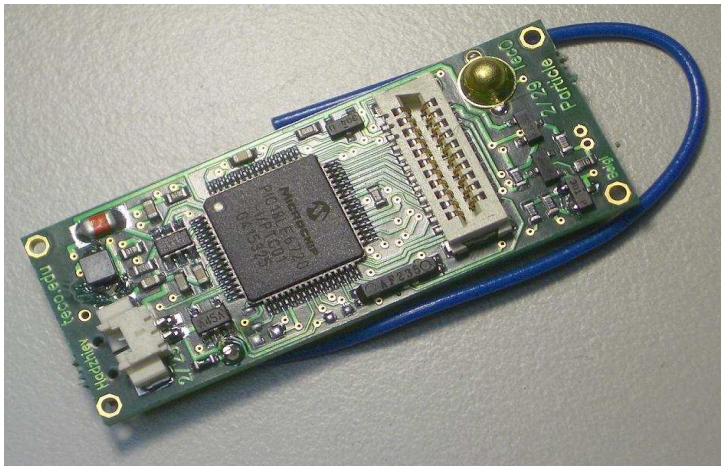
Streams

- Access functions “read”, “write”
- Sequential access
  - enforced by state
- Pass data between streams
  -> Stacking of streams



Hardware

- TecO Particle Computer
- 18f6720 PIC microcontroller
- 128KB FlashROM, 4KB RAM, 512KB external Flash



Resources

- Memory – 512KB Flash
- Power supply – AAA battery
- Wireless communication
- Sensors – acceleration, light, force, temperature, audio, ball switch
- Actuators – LEDs, speaker

Optimization

- Paralallize write operations
  (Flash Buffer, Flash, EEPROM)
  - 13 ms Flash, 4 ms EEPROM, assynchronously programmable
- Mimimize writes on Flash/EEPROM
  - only 50.000-100.000 erase/write cycles per page
- Minimize read times for sensor devices
  - support relatively high sampling rates / bytewise sampling
- Minimize RAM consumption
  - only 4K of RAM available

Performance

Access overhead

	cycles	PIC18F6720
Table look up function	15 cycles	3 µs
Dereferencing state	4 cycles	0.8 µs
Passing Parameters	10 cycles	2 µs
Function pointer call	26 cycles	5.2 µs
Accessing Parameters	10 cycles	2 µs
Writing the buffer	15 cycles	3 µs
Returning	13 cycles	2.6 µs
Overhead of file system read	93 cycles	18.6 µs
Overhead of simple Library call	26 cycles	5.2 µs
Relative overhead	67 cycles	13.2 µs

Performance

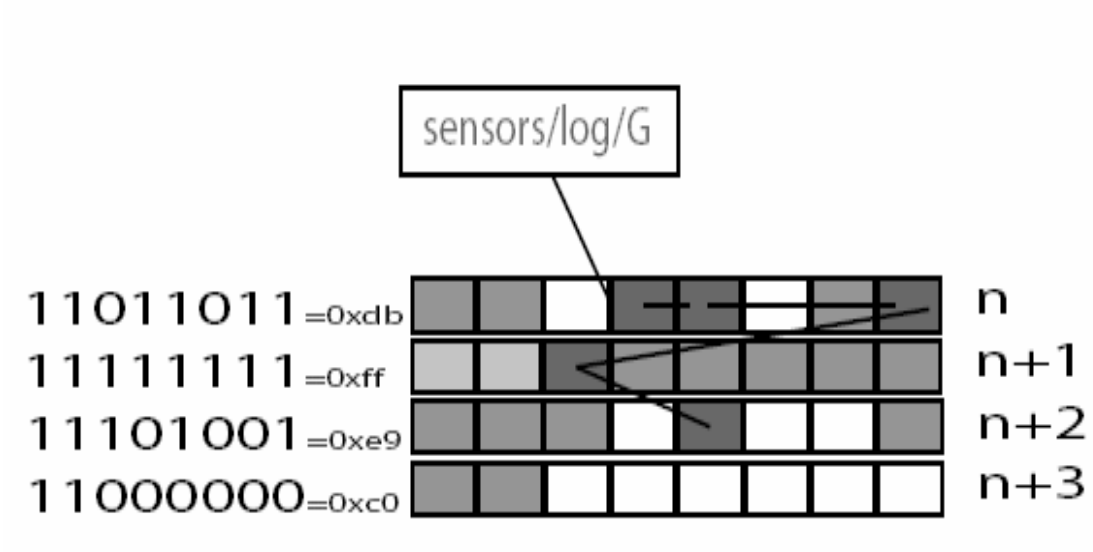
Memory Consumption

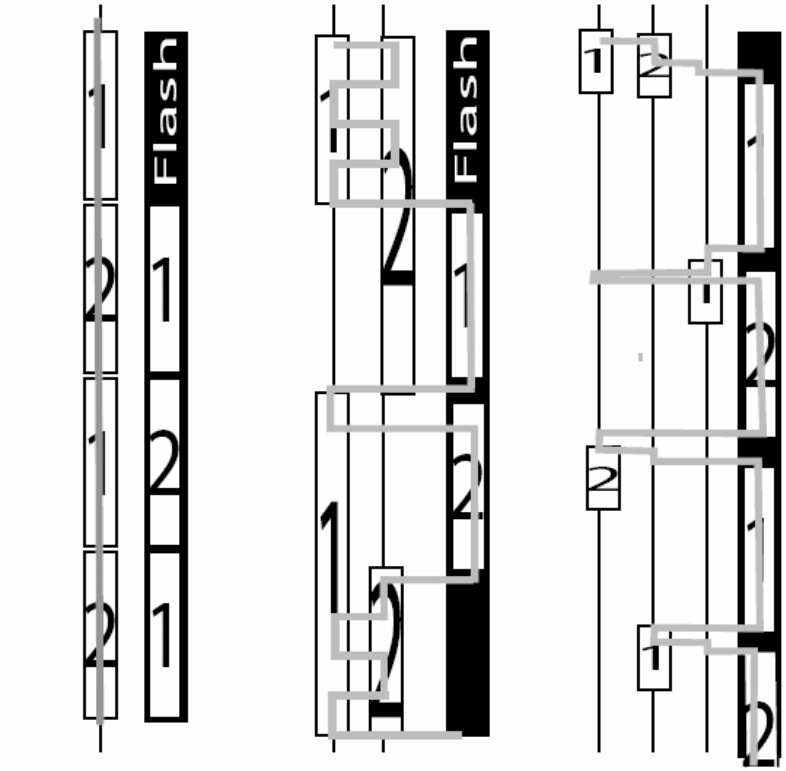
- File library
  - 2 byte on flash per resource (129 per page)
  - 12 byte per file descriptor (3 byte state)
  - Non-Persistent ressources
    - 6 byte + name in RAM

  + Library Code in internal flash
    - about 12 KB (10%)
    - 6K without persistent storage

Integration of persistent storage

- **Simple flash FS**
- **Sequential append-only files**
- **Non-blocking operation**
- **Uncontrolled extrusion**

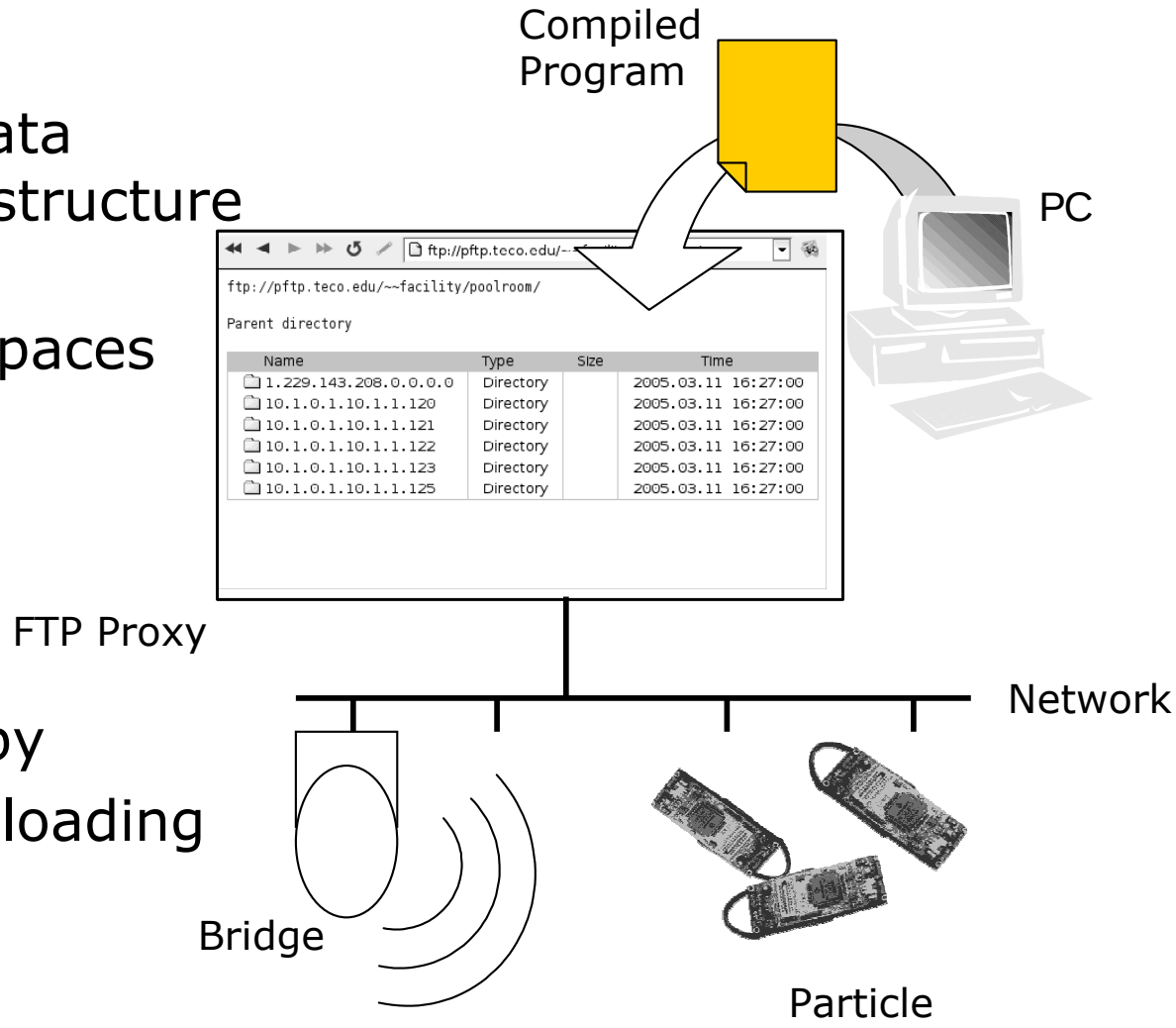




Application - Integration

FTP Proxy

- Seamless transfer of data between backend infrastructure and sensor nodes
- Composition of name spaces using URI scheme



- Programming is file copy
- Sensor logging is downloading

POSIX interface

Applications can be easily ported

libc standard I/O

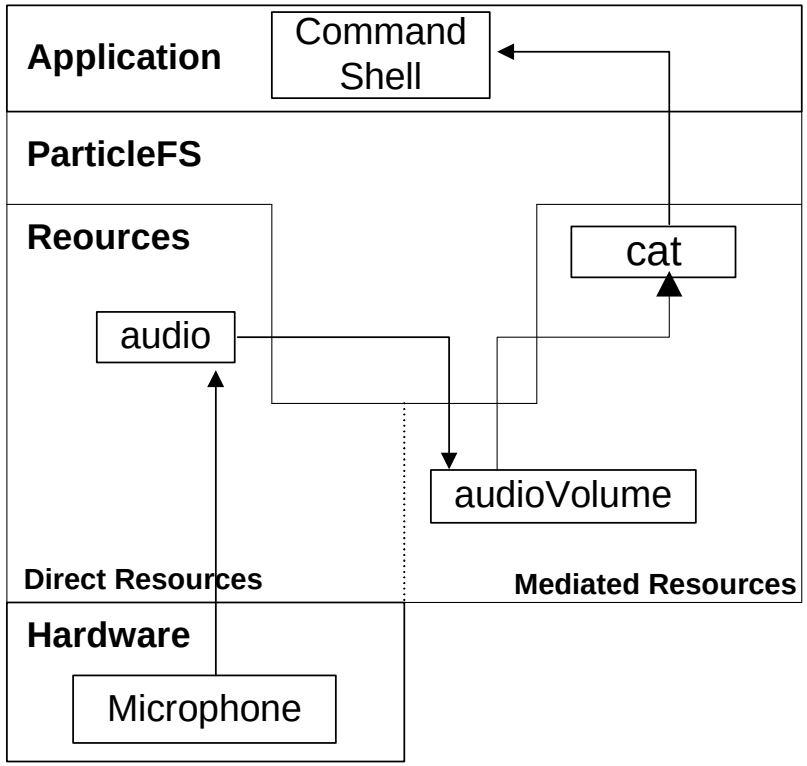
- fprintf
- stdout

compliant C Programming Environment

Application - Shell

File system as interactive runtime environment

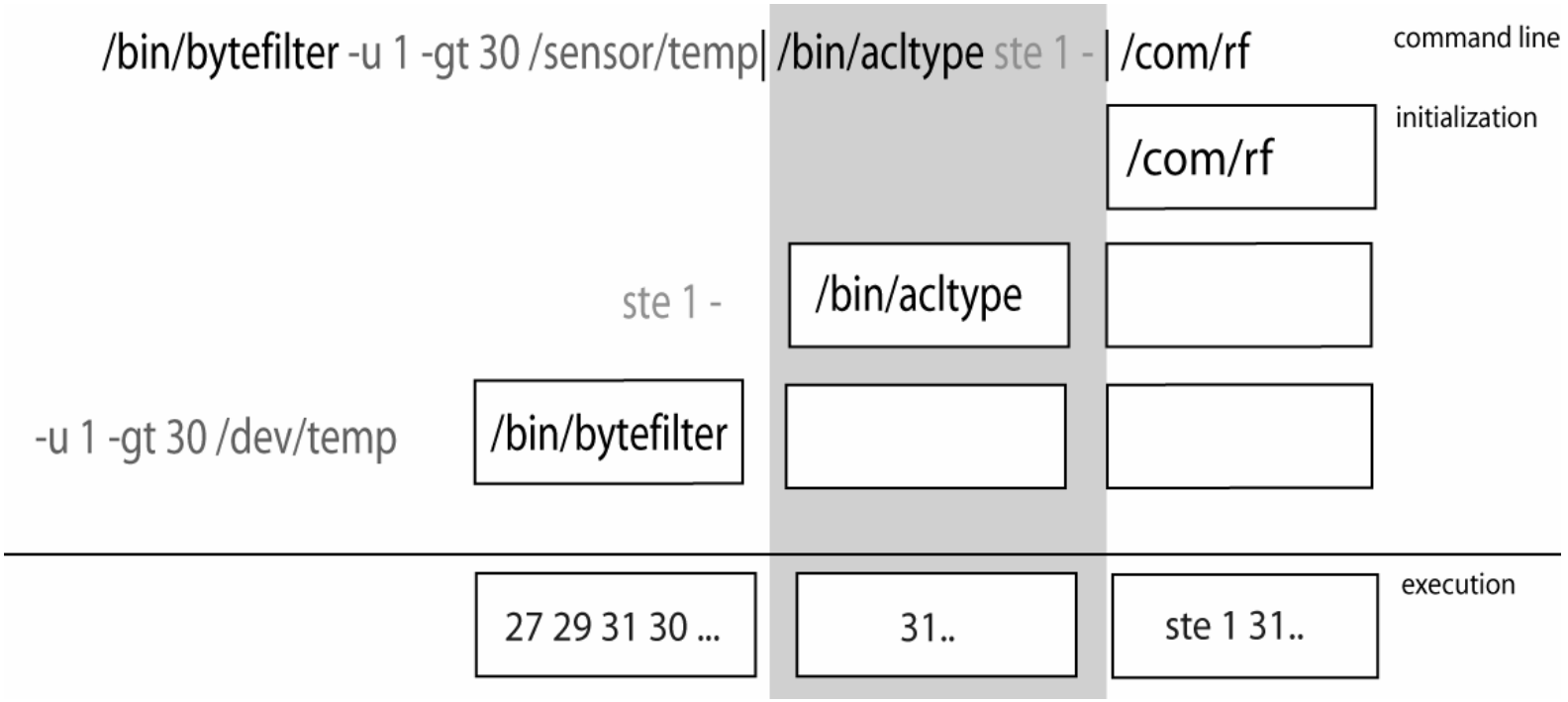
"cat audiovolume"



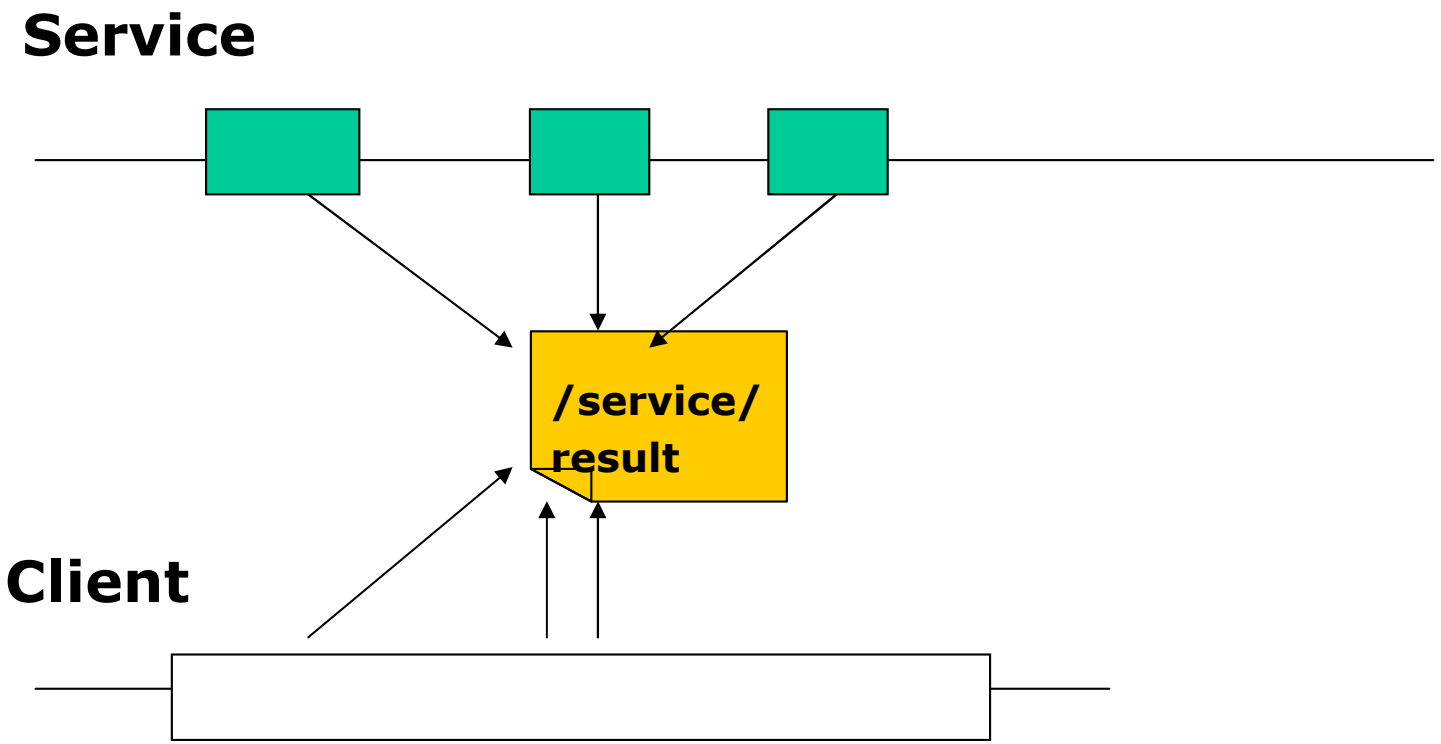
Application - Shell

File system as interactive runtime environment

- Standard output as context
- Popping stack elements from pipe stack



Asynchronous Interaction



Simple IPC mechanism

Dynamic Loadable Modules

- **No linking of function calls needded**

- **Specify fixed location for**
  - file descriptor table
  - open
  - mount

Functionality instantly available after mount

Hot code replacement

Thank you.