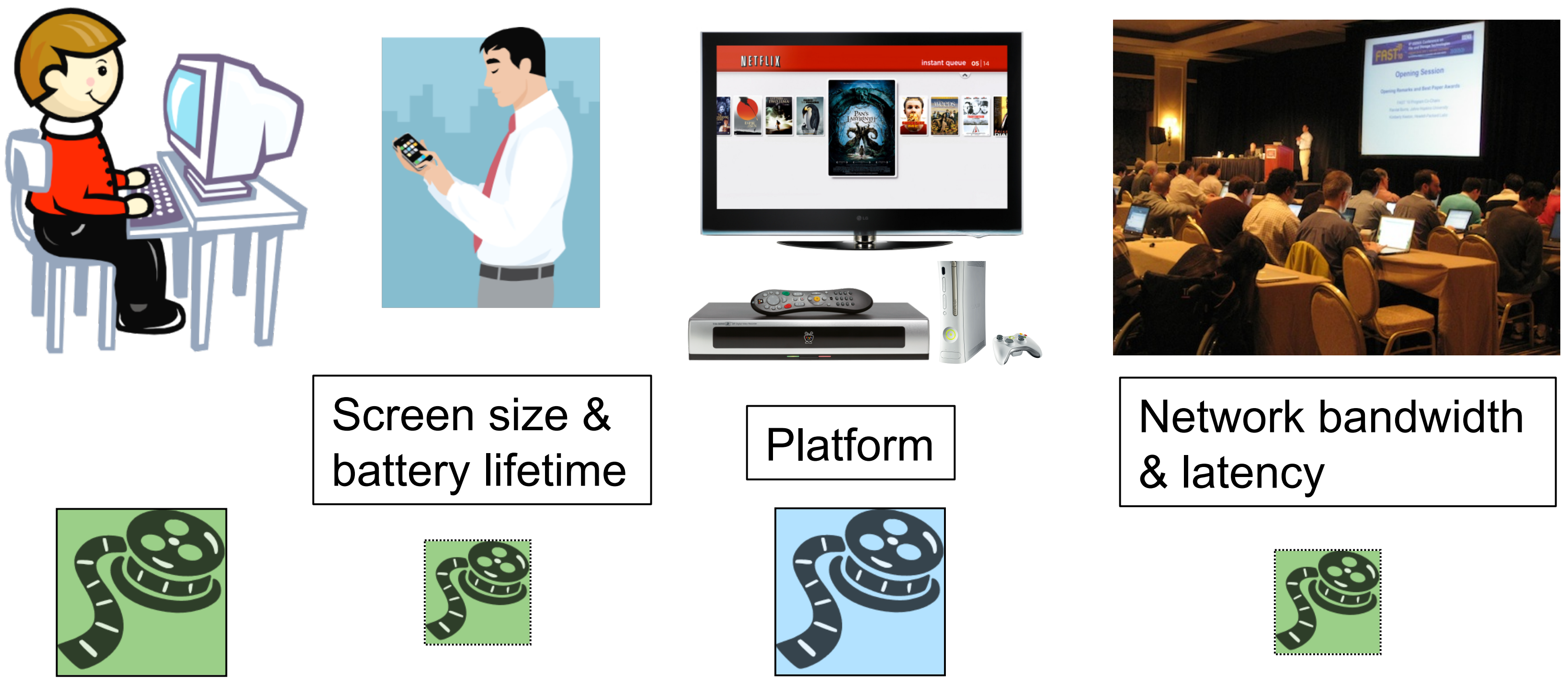
quFiles: The right file at the right time

Kaushik Veeraraghavan
Jason Flinn
Ed Nightingale*
Brian Noble

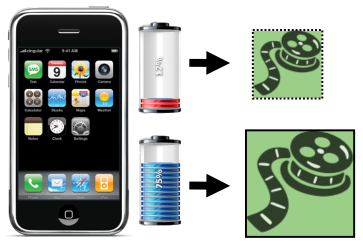
University of Michigan
*Microsoft Research (Redmond)

Users need different data for different contexts

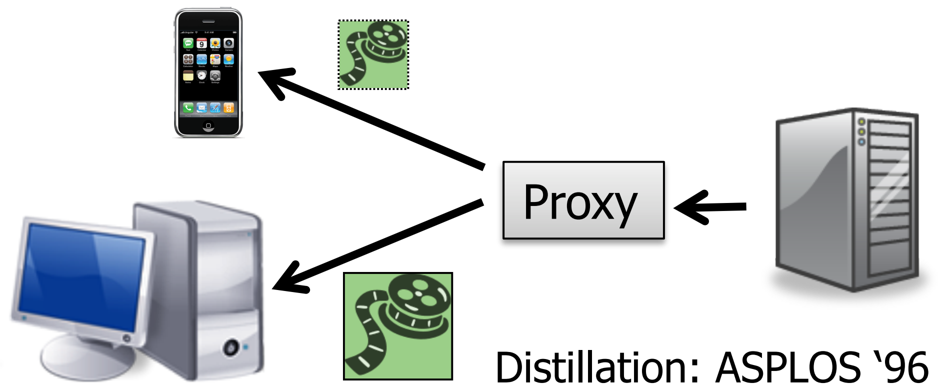


Users want to see the right file at the right time

Decouple adaptation from management



Energy-aware adaptation: SOSP '99



Distillation: ASPLOS '96

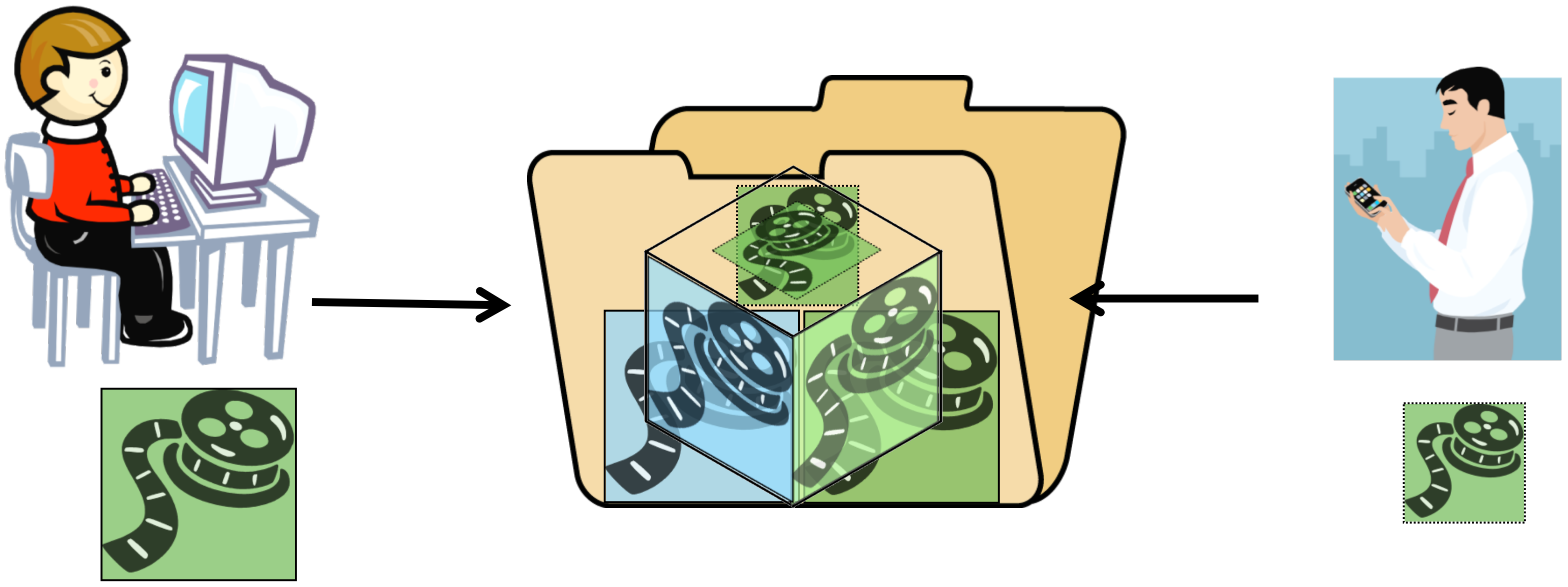
- Problem: each application builds both, an adaptation system and a data management system
- Our contribution: common abstraction for context-aware data management

  Free developers to build interesting adaptation schemes!

Layer context-awareness in existing FS

- The way data is presented to users can be different from how it is stored
  - Change the interface used to access data
- Create new context-aware systems by just writing policies
  - We built two new applications in a couple weeks!
- Existing applications that use the file system become context-aware without any modification

quFiles: a unifying abstraction

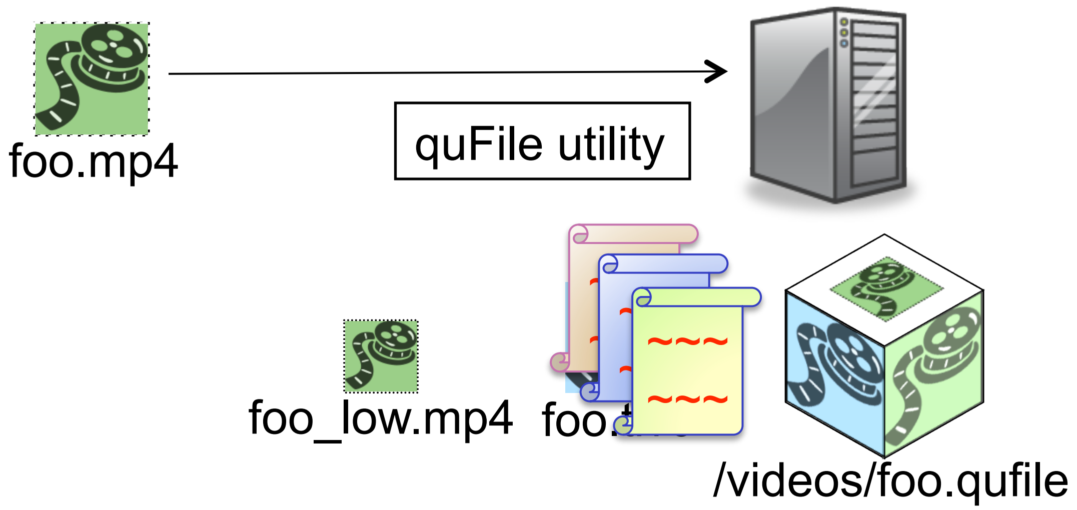


- quFiles multiplex different views of a single logical object
- Context-aware mechanism selects the best representation

Talk outline

- What are quFiles?
- Design & Implementation
- Case studies
- Evaluation
- Related work
- Conclusion

Life of a quFile

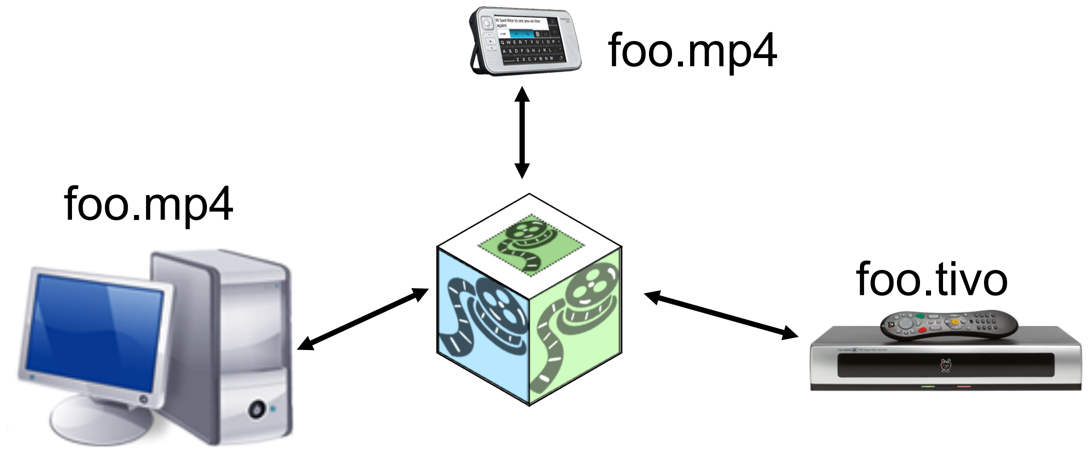

foo.mp4
quFile utility
foo_low.mp4
foo.
~~~
~~~
/videos/foo.qufile

- Utility creates alternate representations of video
- Utility creates a quFile and moves representations into it
- Links in the policies

Name policy: choosing the right name

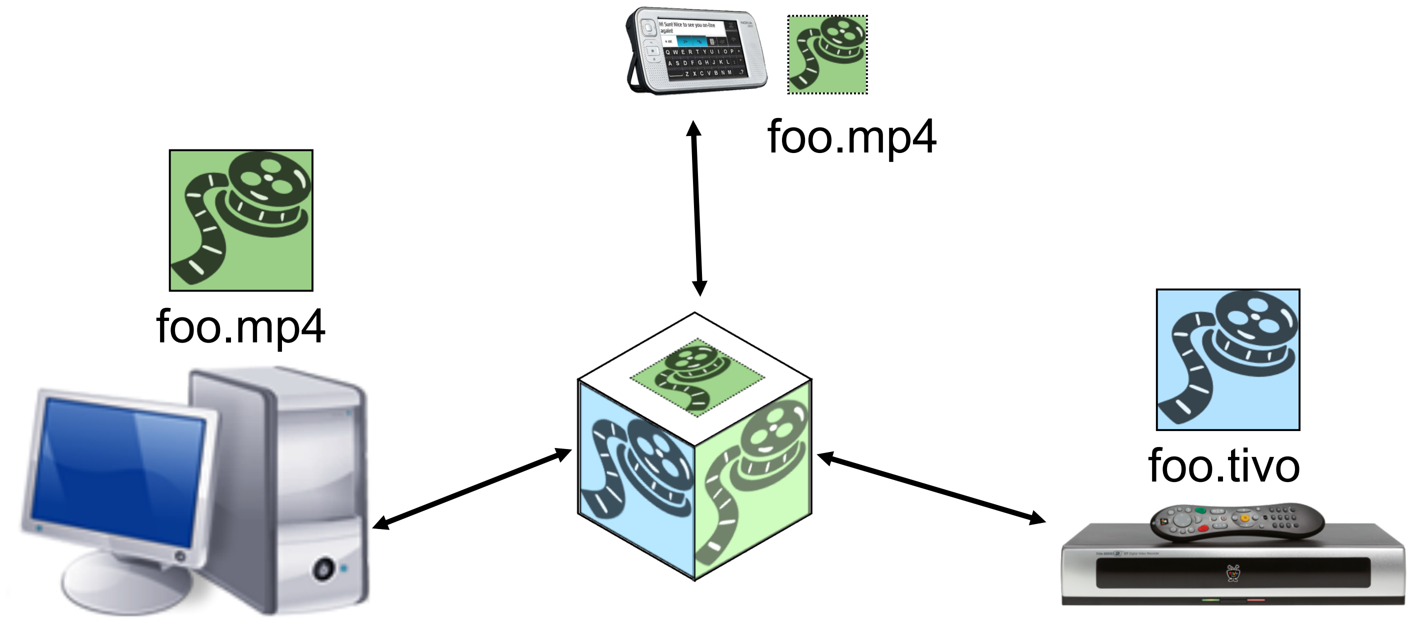
Name policy:

```
If (device == TiVo) {
  return "foo.tivo";
} else {
  return "foo.mp4";
}
```



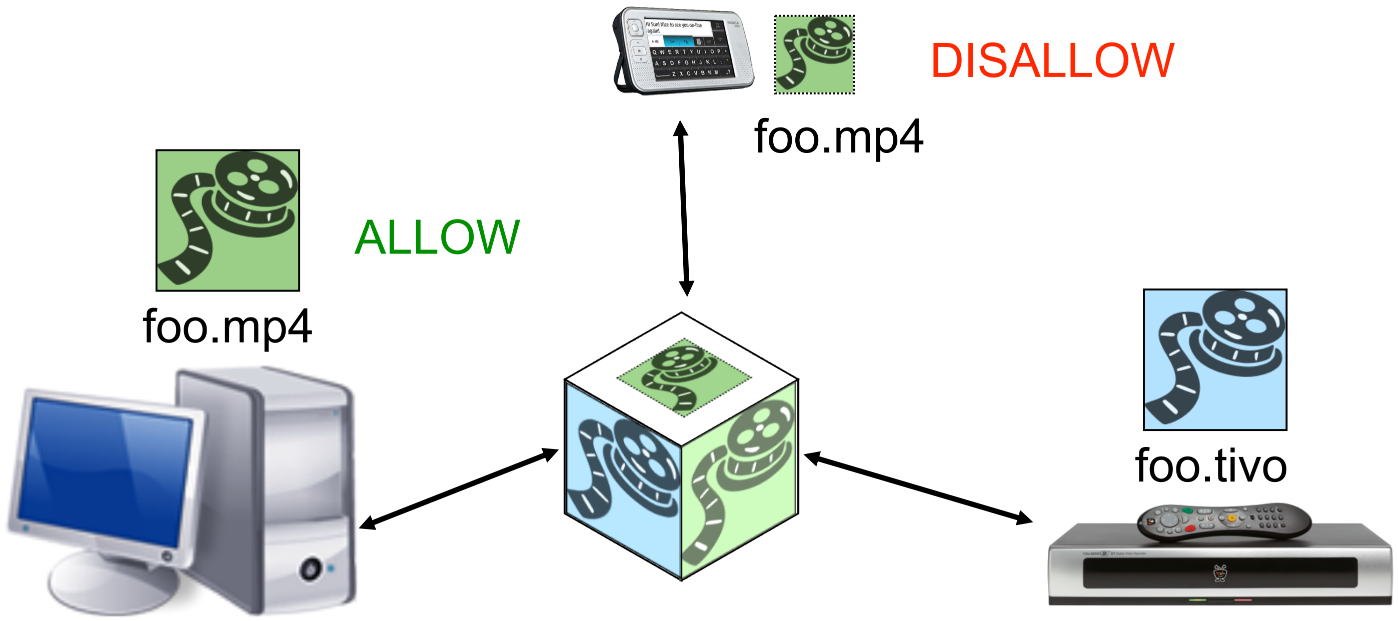
- Name policy: 0 or more file names
- Policy may dynamically instantiate a new name

Content policy: choosing the right content



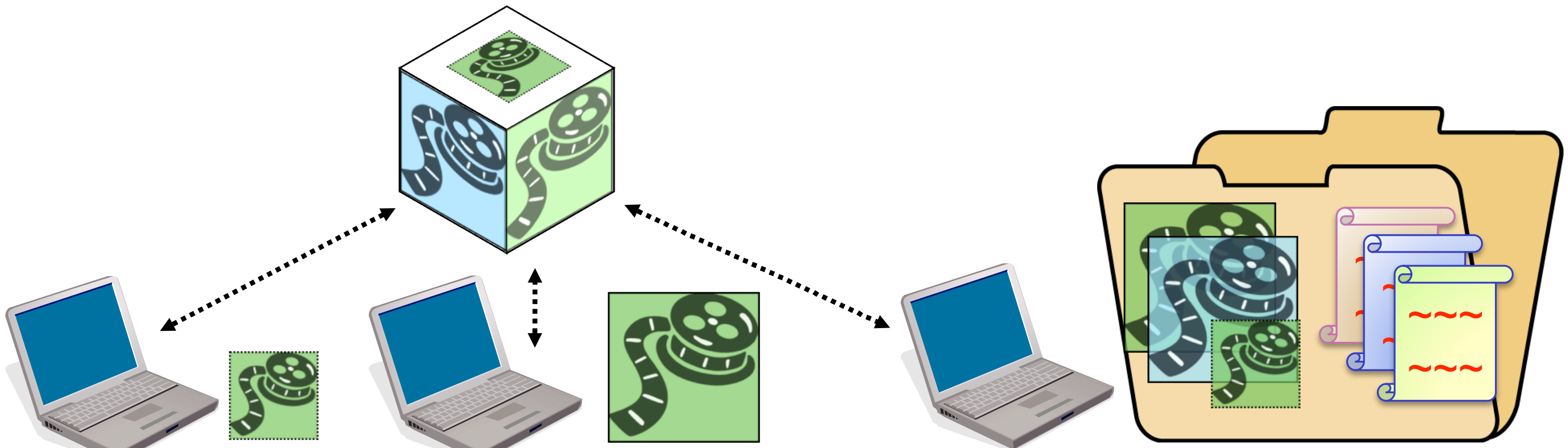
- Content policy: specific content for file name
- Policy may dynamically create a new file and content

quFile edit and cache policies



- Edit policy: allow, disallow or version
- Cache policy: which representation to cache

quFiles support multiple views



```
% ls video.qufile.highres
%
```

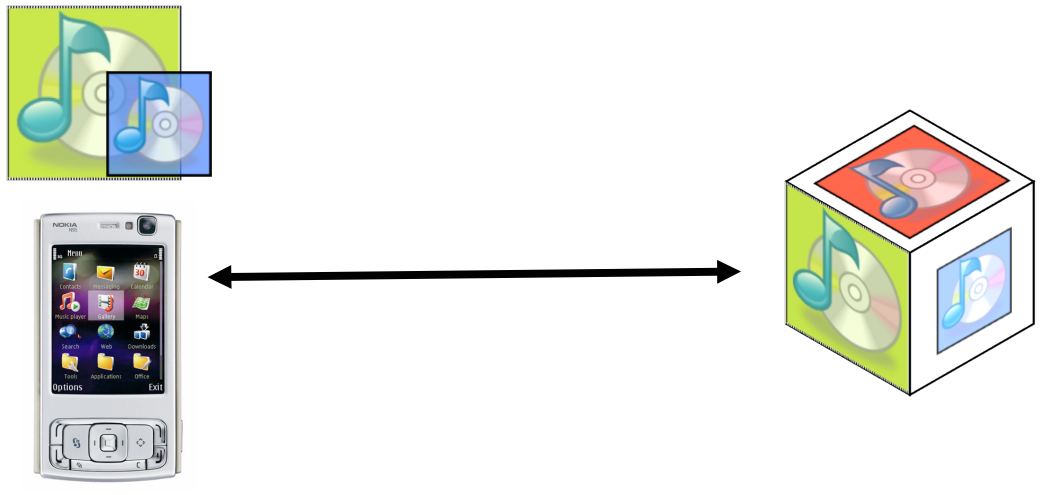
```
% ls video.qufile.raw
%
```

- Raw view: shows all contents i.e. representations, policies,…
- Custom view: policy may return any representation it wishes
- No application modification is required to see other views

Talk outline

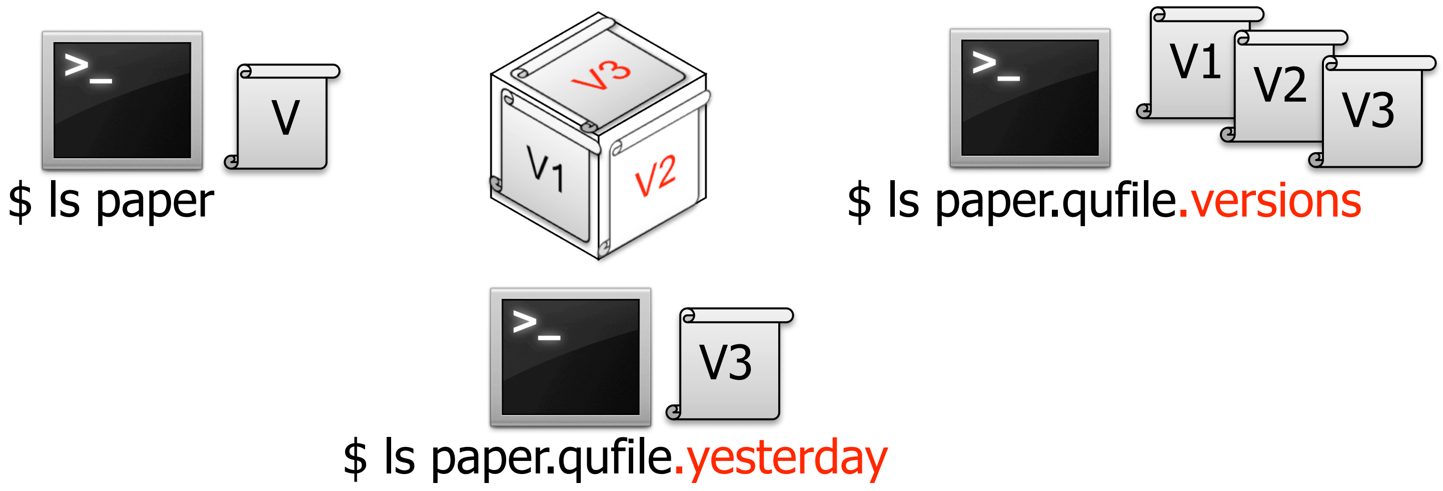
- What are quFiles?
- Design & Implementation
- Case studies
- Evaluation
- Related work
- Conclusion

Power management



- Cache policy: use spare storage to cache WAV
- Name & content policy: return WAV if cached, else mp3
- 4-11% battery lifetime gain; lines of policy code: 94

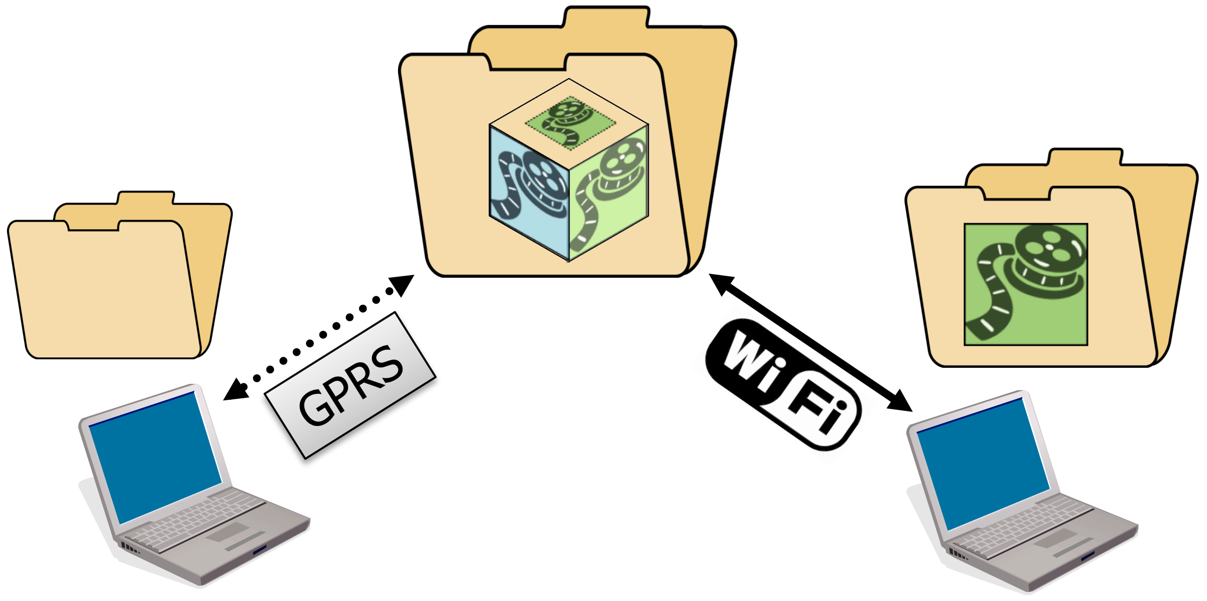
Copy-on-write versioning



```
$ ls paper
$ ls paper.qufile.versions
$ ls paper.qufile.yesterday
```

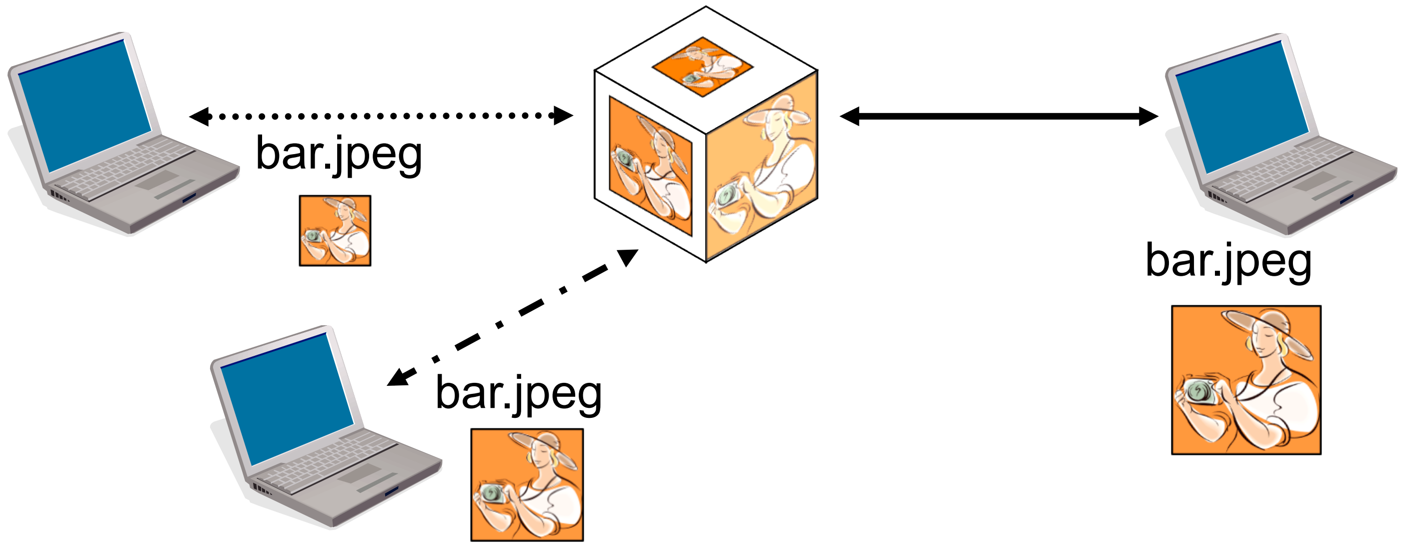
- Edit policy: save information to an undo log
- Custom versions view
  - Name policy: returns names of all past versions (1, 2 or more)
  - Content policy: dynamically generates past version
- Lines of policy code: 55

Resource-aware directory listing



- Default view: list files viewable given network quality
- Custom “all” view: “currently_unplayable” suffix
- Lines of policy code: 98

Application-aware adaptation: Odyssey



- Name: bar.jpeg to all clients
- Content: best image served in 1 second
- Edit: disallows content writes, allows metadata writes
- Lines of policy code: 82

Talk outline

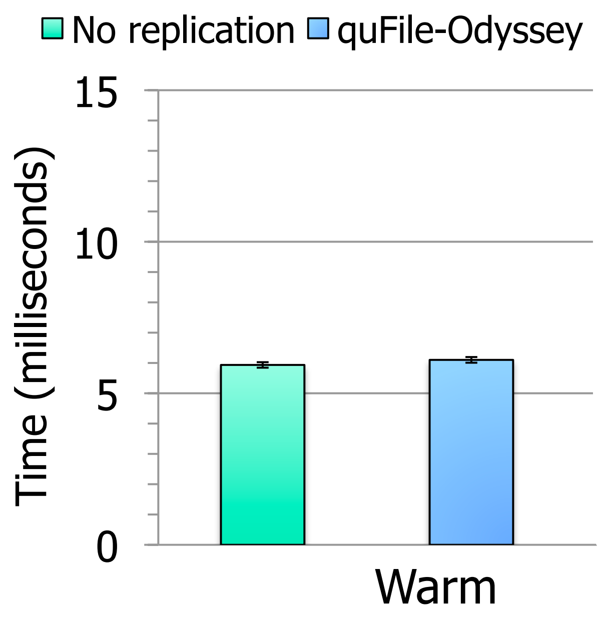
- What are quFiles?
- Design & Implementation
- Case studies
- Evaluation
- Related work
- Conclusion

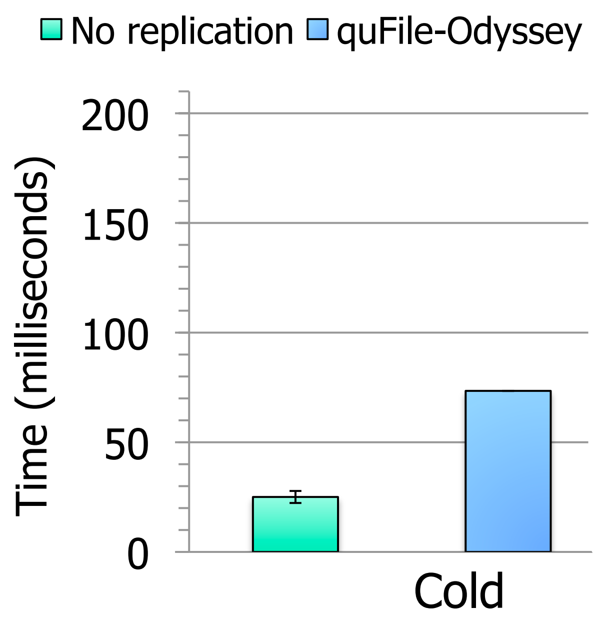
quFiles are easy to implement and use

- quFiles are easy to incorporate in a file system
  - quFiles add 1,600 lines to BlueFS's 28,000.
- Almost all policies (see table) require less than 100 lines.
  - Each case study in a week or two. Some 1-2 days.

Component	**Name**	**Content**	**Edit**	**Cache**	**Total**
Power Management	32	18	8	36	94
Copy-on-write versioning	29	18	8	N/A	55
Security	20	33	8	N/A	61
Resource-aware directory listing	64	26	8	N/A	98
Odyssey	23	27	32	N/A	82
Platform spec. video display	31	30	8	43	112

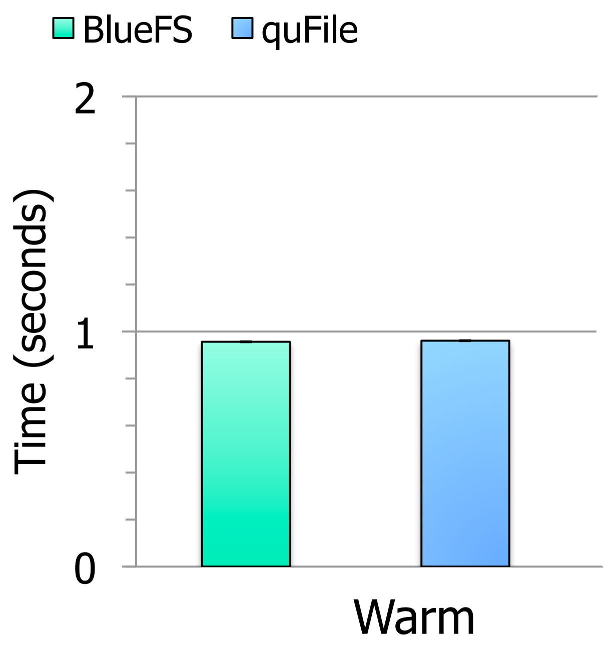
Micro-benchmark: Directory listing overhead

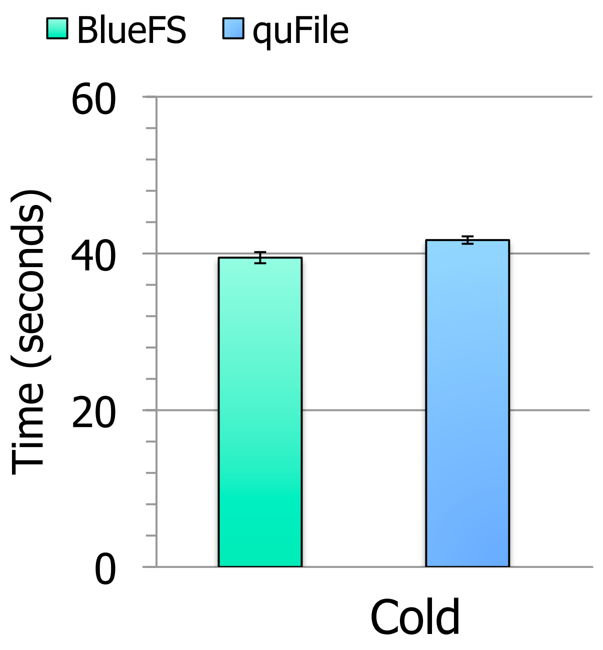




- Worst-case quFile overhead as there's no activity to amortize cost
  - Only 3% overhead for warm; 0.5 ms overhead per file for cold
- quFiles are 2X-3X better than Replication

Kernel grep





- grep Linux 2.6.24 source: grep –Rn “foo” linux (9 occurrences)
- 1% overhead for warm; 6% overhead for scenario
- Search all versions: grep –Rn “foo” linux.qufile.versions (18 occurrences)
    - 2X overhead in warm; 31% in cold case

Related work

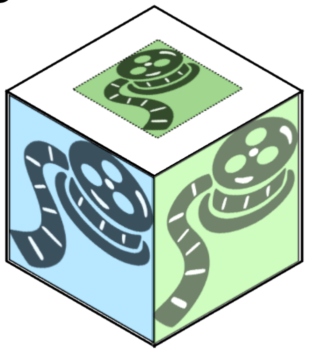
- Semantic File System
  - Only expands name space but not content
- Adaptation systems: Ninja, Odyssey, Puppeteer, …
  - No application or OS modification, no proxy. Adaptation policies.
- Partial-replication: Cymbiosis, PRACTI, Perspective
  - Filter-based caching policies can be augmented with context
- Dynamic resolution of file content: OS X bundles, AFS @sys
  - General abstraction w/o baking resolution policies in FS
- Materialized views in databases
  - Context-aware generation of views; operate on data without schema

Conclusion

- quFiles provide first-class support for context in file systems
  - Multiplex different views onto single logical object
  - Context-aware policies select the best view

- Context-aware systems can be easily built by simply providing quFile policies

- Thank you!



Building blocks of quFiles

- Policies are file system extensions
  - User-level software fault isolation is fine
- File system change notifications
  - To trigger quFile utilities (automation)
- File system should support directories
- Context library
  - Simple to build: ours is ~250 LOC

Why put quFiles in the file system and not middleware, library, …

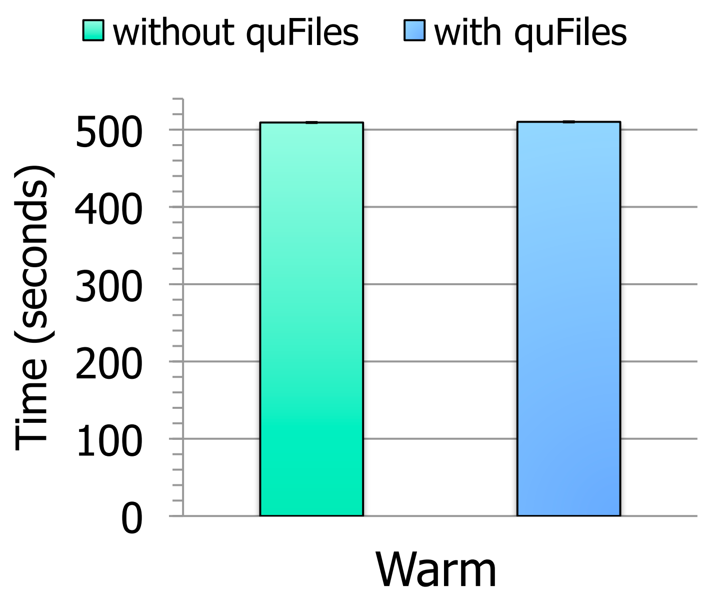
- Any application that uses the file system now becomes context-aware
  - Transparency ensures backward compatibility
- quFiles are a simple abstraction in the FS
  - Hooking into POSIX API is simple
  - readdir, lookup, commit_write, unlink, rename

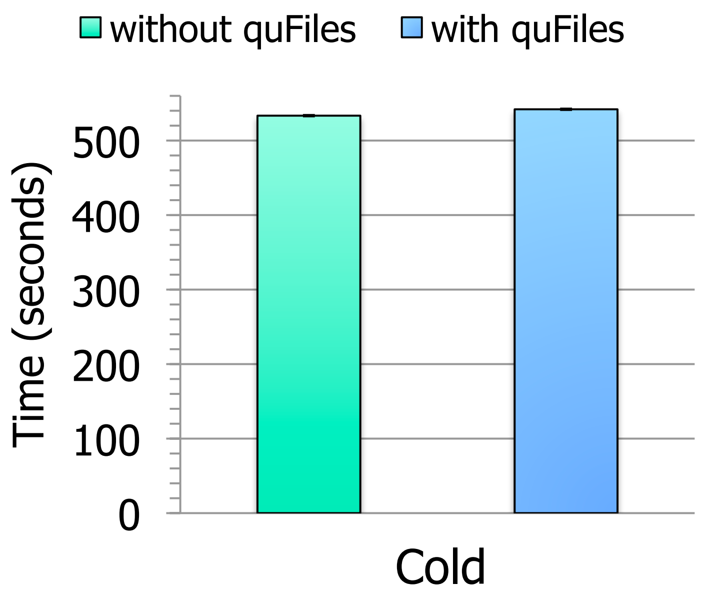
Case study applicability

- Local & Distributed file systems
  - Resource management
    - E.g.: if battery is low, display low-res video
  - Copy-on-write versioning
  - Context-aware redaction

- Distributed file systems
  - Resource-aware directory listing
  - Application-aware adaptation: Odyssey

Andrew-style make





- Make Linux 2.6.24 kernel
  - quFile: version all source files (.c, .h or .S) – 19,844 of 23,062 files
- Negligible overhead for warm, 1% overhead for cold scenario